BBN 6473

THE SIMNET MANAGEMENT, COMMAND AND CONTROL SYSTEM

(2)

31 March 1987

BBN Laboratories Incorporated

Contract MDA903-83-C-0168

SIMNET

DARPA
LT. COL JACK A. THORPE
(202) 694-8232, AUTOVON 224-8232
TACTICAL TECHNOLOGY OFFICE
DEFENSE ADVANCED RESEARCH PROJECT AGENCY (DARPA)
1400 WILSON BLVD.,ARLINGTON, VA 22209-2308

CONSULTANT
GARY W. BLOEDORN, COL (USA RET)

PERCEPTRONICS
MR. RICHARD TAYLOR
(818) 884-3485
21122 ERWIN ST, WOODLAND HILLS, CA 91367-3713

BBN LABORATORIES INCORPORATED
DR. DUNCAN C. MILLER
(617) 497-3334
10 MOULTON STREET, CAMBRIDGE MA 02238

BBN DELTA GRAPHICS INCORPORATED
MR. MICHAEL CYRUS
(206) 746-6800
14100 S E 36th ST, BELLEVUE, WA 98006

Report No. 6473

The SIMNET Management, Command and Control System

Contract MDA903-83-C-0168

Arthur R. Pope
Tim Langevin
Andrew R. Tosswill

March 1987

Prepared by:

BBN Laboratories Incorporated
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
Tactical Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

Preface

SIMNET: Advanced Technology for the Mastery of War Fighting

SIMNET is an advanced research project sponsored by the Defense Advanced Research Projects Agency (DARPA) in partnership with the United States Army. Currently in its third year, the goal of the program is to develop the technology to build a large-scale network of interactive combat simulators. This simulated battlefield will provide, for the first time, an opportunity for fully-manned platoon-, company-, and battalion-level units to fight force-on-force engagements against an opposing unit of similar composition. Furthermore, it does so in the context of a joint, combined arms environment with the complete range of command and control and combat service support elements essential to actual military operations. All of the elements that can affect the outcome of a battle are represented in this engagement, with victory likely to go to that unit which is able to plan, orchestrate, and execute their combined-arms battle operations better than their opponent. Whatever the outcome, combat units will benefit from this opportunity to practice collective, combined arms, joint war fighting skills at a fraction of the cost of an equivalent exercise in the field.

While simulators to date have been shown to be effective for training specific military skills, their high costs have made it impossible to buy enough simulators to fully train the force. Further, because of the absence of a technology to link them together, they have not been a factor in collective, combined arms, joint training. SIMNET addresses both of these problems by aiming its research at three high payoff areas:

- Better and cheaper collective training for combined arms, joint war fighting skills
- A testbed for doctrine and tactics development and assessment in a full combined arms joint setting
- A "simulate before you build" development model

These payoffs are achievable because of recent breakthroughs in several core technologies which have been applied to the SIMNET program:

- High speed microprocessors
- Parallel and distributed multiprocessing
- Local area and long haul networking
- Hybrid depth buffer graphics

- Special effects technology
- Unique fabrication techniques

These technologies, applied in the context of "selective fidelity" and "rapid prototyping" design philosophies, have enabled SIMNET development to proceed at an unprecedented pace, resulting in the fielding of the first production units at Fort Knox, Kentucky, just three years into the development cycle.

In addition to the basic training applications, work is underway to apply SIMNET technology in the area of combat development to aid in the definition and acquisition of weapon systems. This is made possible because of the low cost of the simulators, the ease with which they can be modified, and the ability to network them to test the employment of a proposed weapon system in the tactical context in which it will be used, i.e., within the context of the combined arms setting.

Work on SIMNET is being carried out by co-contractors Bolt Beranek and Newman, Inc. (BBN) and Perceptronics, Inc. Perceptronics is responsible for training analysis, overall system specification, and the physical simulators, and BBN is responsible for the data communication and computer-based distributed simulation and the computer image generation (CIG) subsystems. The project is a total team effort.

DARPA is the DoD agency chartered with advancing the state of the art in military technology by sponsoring innovative, high risk/high payoff research and development.

Table of Contents

Table of Illustrations

1. INTRODUCTION

The SIMNET project has resulted in the development of a network of interactive combat simulators for vehicles such as the M1 Abrams main battle tank. These simulators operate in the context of a joint, combined arms environment that includes the command and control, combat support, and combat service support elements essential to actual military operations. The SIMNET *Management, Command and Control (MCC)* system is a collection of hardware and software that creates this battlefield environment.

1.1. Purpose of the MCC System

The overall purpose of the MCC system in a simulation is twofold: to aid in setting up the simulation, and to simulate certain elements of the battlefield environment.

The MCC system is designed to serve a unit as large as a battalion or task force. Thus it simulates those elements of the battlefield environment (apart from the combat vehicles themselves) that correspond to the assets available to a single battalion. Larger operations are possible with SIMNET; in an exercise involving multiple battalions, each battalion is served by its own MCC system.

The MCC system described here is to be used in support of a maneuver force, so the things it simulates include elements that are part of the typical maneuver battalion, such as supply trucks and mortars. Also simulated are elements typically made available in support of the battalion, such as howitzers and close air support aircraft. Altogether, these simulated elements include the following:

- a tactical operations center (TOC), which is part of the battalion command post
- an administration and logistics (admin/log) center, from which the battalion's combat service support is coordinated
- howitzers and mortars capable of delivering indirect fire in support of the battalion's combat units
- aircraft performing close air support missions
- supply depots and trucks capable of resupplying fuel and ammunition to combat vehicles
- maintenance teams capable of recovering disabled combat vehicles and performing certain repairs to them

It is intended that the MCC system operate under the direction of the appropriate battalion staff members in simulating each of these elements. The simulated mortars, for example, are controlled by the battalion's fire support officer and the simulation of close air support is controlled by the battalion's air liaison officer. The ammunition and fuel trucks move under the direction of the battalion's support platoon leader, and recovery and repair of combat vehicles are directed by the battalion maintenance officer or his staff. The MCC system includes individual consoles through which each of these participants controls the simulated elements for which he is responsible.

The MCC system's simulation of each element is performed to an appropriate level of detail as dictated by the overall SIMNET objectives. The simulated mortars, for example, are present on the battlefield, where they are visible to combat vehicle crewmembers and vulnerable to enemy action. Moreover the mortars can deliver fire that not only can be seen, heard, and felt from nearby combat vehicles, but also can disable or destroy those vehicles. However the mortars are not driven about the battlefield or loaded and fired by individual human crews. The simulation of the entire mortar platoon simply responds to orders issued directly by the fire support officer. This level of detail is appropriate because, whereas the effect those mortars can have on the outcome of the battle is relevant to the SIMNET objective of training combat vehicle crews, the training of crews to operate the mortars themselves is not a SIMNET objective.

Simulating certain elements of the battlefield environment is one purpose of the MCC system. Its other purpose is to allow a simulation to be established.

The MCC system is designed to be compatible with the maneuver battalion's usual procedures for planning and conducting an operation. Typically, a higher headquarters will impose constraints or a scenario within which the operation must be performed. Working from these constraints, the battalion staff plan their operation. The products of this planning and the overall constraints for the operation are supplied to the MCC system to establish the initial parameters of a new simulation exercise. This information includes:

- how the simulated combat vehicles are organized into platoons and companies, and, if a force–on–force exercise is being conducted, how the vehicles are divided into opposing forces. This information is supplied to the MCC system by the battalion S3 (operations) officer.

- the location of each combat vehicle at the beginning of the exercise, and the quantities of ammunition and fuel on board each vehicle. Each company or platoon leader may supply this information for the vehicles in his unit.

- the locations of the TOC, admin/log center, resupply trucks, maintenance teams, howitzers, and mortars at the beginning of the exercise. This information is supplied to the MCC system by the various battalion staff members responsible for planning the deployment of those elements.

- the distribution of ammunition and fuel to each supply truck, as determined by the support platoon leader. He works within the constraints imposed upon the operation as to the total quantities of fuel and ammunition available.
- the quantities of ammunition stocked with the howitzers and mortars, as reported by the fire support officer.
- the number of sorties of close air support aircraft available, as reported by the air liaison officer.
- lists of preplanned indirect fire targets and missions prepared by the fire support officer.
- a list of preplanned close air support missions prepared by the air liaison officer.

This information establishes only the initial parameters of the exercise. As the exercise proceeds, most of these parameters, such as the locations of vehicles, can and will be changed by the human participants in the exercise. The original constraints imposed by the higher headquarters, however, remain fixed.

One individual is unique among all those participating in a simulation exercise in that he is the only one performing a role in the exercise not related to the function he performs during an actual operation. All other participants, including both combat vehicle crew and battalion staff members, perform roles in the simulation that are derivative of the roles they perform in combat. The one exception is called the *BattleMaster*. Using the MCC system, the BattleMaster performs a variety of administrative tasks in order to provide the correct organizational environment in which the battalion may conduct its simulation exercise. An example of these tasks is the handling of requests from the fire support officer to have the howitzers moved on the battlefield, thus simulating the role of the direct support artillery battalion commander and staff. Another example is the handling of requisitions from the support platoon leader for additional supplies of ammunition or fuel. It should be noted that in performing these duties the BattleMaster serves only as a coordinator, not as a controller or umpire for the exercise.

1.2. Implementation of the MCC System

The MCC system is distributed over a collection of several computers that are interconnected by an AppleTalk® network. One of these computers, a Masscomp® 5600 workstation, controls the system. This computer is referred

® Apple and AppleTalk are registered trademarks of Apple Computer, Inc.

® Masscomp is a registered trademark of the Massachusetts Computer Corp.

to as the *MCC host*. The other computers are Apple® Macintosh™ personal computers. Most of these Macintoshes serve as consoles through which the various users of the MCC system may interact with it. One Macintosh is used as a bridge between the the MCC host and the AppleTalk network.

In all of its activities the MCC system must work closely with other parts of the overall SIMNET system, especially the combat vehicle simulators. The MCC system communicates with the rest of the SIMNET system through its attachment to the SIMNET local area network. The MCC host is the point of attachment to that network.

Colonel Gary Bloedorn, (USA Ret.), determined the original requirements for the MCC system and continued to guide its form throughout development. The system was developed at BBN Laboratories Incorporated by a team that at various times included Arthur Pope, Tim Langevin, Andrew Tosswill, Brian Vaughan, Phillip Yoo, and James Rayson. Software was incorporated from other areas of SIMNET development, including that produced by Daniel Van Hook, Alan Dickens, and David Epstein.

We are grateful for the help of many individuals who reviewed preliminary versions of the MCC system, and offered valuable suggestions for its improvement.

The MCC system described in this document is operational, and in daily use as part of the SIMNET installation at the Ft. Knox Armor Center. The system is undergoing continuous, evolutionary improvement. This document describes the system as it existed in the fall of 1986.

1.3. About this Document

The primary purpose of this document is to describe how the MCC system is implemented. A secondary purpose is to provide explanations for some of the design decisions made during implementation.

Any reader seeking a detailed understanding of the MCC system from this document should:

- understand something of how an armor battalion organizes and operates. The U.S. Army manual entitled *The Division 86 Tank Battalion/Task Force*[1] contains much information on this topic.
- be familiar with how the MCC system is used by the battalion staff and the BattleMaster. Sections of *The SIMNET Master Documentation*[2] cover these topics.

™ Macintosh is a trademark of Apple Computer, Inc.

- understand how the MCC system interacts with other SIMNET components, as explained in *The SIMNET Network and Protocol*[3].

In this chapter we have presented a brief overview of the MCC system, focusing on what it does and who uses it. We describe the functions of the MCC system in more detail in Chapter 2. Chapter 3 describes the architecture of the system in terms of its major hardware and software components, the purposes of those components, and the relationships between them. The AppleTalk network, which plays a central role in the operation of the MCC system, is discussed in Chapter 4. Chapters 5 and 6 describe in more detail the various software components of the system. The Appendices contain summaries of the parameters that tailor the specific behavior of the MCC system. and the nature of the communication between the system's software components.

2. FUNCTION

This chapter summarizes the behavior of the MCC system, emphasizing the functions that the system performs and the things it simulates.

Three key principles have guided the specification of what the MCC system should do:

- The MCC system must provide the support necessary for combat vehicle crewmembers to carry out certain training tasks. Thus it provides both help in setting up a simulation exercise, and simulations of the battlefield phenomena relevant to the kinds of training to be performed.

- The MCC system must conform to standard operating procedures. Users of the MCC system will be armor battalion personnel, and they will perform roles in the simulation exercise that are related to the actual responsibilities they have during combat. Thus what the MCC system does for each of its users is directly related to what that user does in combat.

- The MCC system must be easy to learn and natural to use, so that the battalion personnel who operate it can use it with minimal additional training. Thus, in communicating with the MCC system, these personnel use the military concepts and terminology with which they are already familiar.

We will describe here how the MCC system is typically used by a maneuver battalion for an exercise involving much of the battalion staff as well as four companies of combat vehicles. However nothing prevents the MCC system from being used for a much smaller exercise with just a few combat vehicles participating, or even none at all.

In this chapter we have included several diagrams showing the relative locations of vehicles placed on the battlefield by the MCC system. Figure 2-1 contains a key to the symbols used in these diagrams for representing various kinds of vehicles.

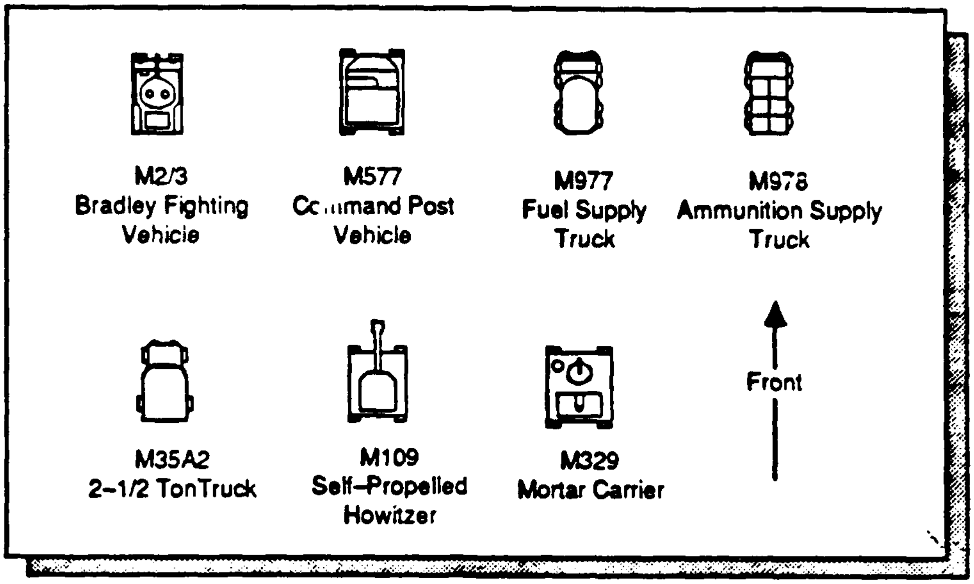


Figure 2-1. Key to vehicle type symbols.

2.1. Consoles

Several consoles are included in the MCC system for use by the BattleMaster and different members of the battalion staff. Each console serves a specific purpose.

- The *SIMNET Control Console* is first used by various members of the battalion staff to establish the simulation exercise, and then, while the exercise is underway, it is used by the BattleMaster to perform his administrative tasks.
- The *Admin/Log Console* is used by the support platoon leader to control the battalion's simulated ammunition and fuel supply trucks.
- The *Maintenance Console* is used by maintenance personnel, such as the battalion maintenance officer, to control the battalion's simulated maintenance teams. It is also used to perform repairs on combat vehicles, and to recover (i.e., tow) combat vehicles across the battlefield.
- The *Fire Support Console* is used by the fire support officer to provide indirect fire support. With this console he issues orders for fire missions to the simulated mortars and howitzers, and controls the placement of the mortars.
- The *Close Air Support Console* is used by the air liaison officer to provide air support by directing the aircraft to their targets on the battlefield.

Of all these consoles, only the SIMNET Control Console is active, or available for use, at the beginning of the exercise. The other consoles become active once the parts of the simulation they control have been initialized from the SIMNET Control Console.

2.2. Computer-Controlled Vehicles

The MCC system simulates a number of vehicles that appear on the battlefield, such as ammunition trucks and self-propelled howitzers. These are called *computer-controlled vehicles (CCVs)* to distinguish them from the combat vehicles controlled by individual crews in simulators. The name CCV is really a bit of a misnomer, for these vehicles are still ultimately controlled by people, not computers. People generally control the CCVs by telling them where to go, and when.

All of the computer controlled vehicles share certain characteristics. They can be moved about anywhere on the battlefield while always travelling at 25 kilometers per hour, but they are never visible while en route. They simply disappear at the moment they are dispatched from one location, and reappear at the moment they arrive at their destination. When a CCV arrives, the MCC system always takes care to place it on a patch of ground that is navigable, and not on top of an obstacle such as a house or another vehicle.

In order to place CCVs, among other things, on the ground, the MCC system has a *terrain database* of information about the battlefield. The terrain database specifies the elevation, slope, and type of soil at each point on the battlefield. It also specifies the shapes and locations of fixed objects, such as houses, that are on the battlefield. As many as three different terrain databases may reside on the MCC system, each representing an area as large as 50 by 50 kilometers. For any exercise, one of these terrain databases is chosen as the battlefield upon which the exercise will take place.

CCVs are not restricted to travelling only within the bounds of the terrain database, and situations do arise where they must travel outside those bounds. For example, the simulated supply depots are usually established some distance from the battle area so that the supply trucks have a realistic distance over which they must transport supplies. As a result, the depots may be completely outside the area covered by the terrain database. In this case supply trucks may reach those depots, but they will not be visible while doing so.

A CCV is destroyed when it is struck by direct fire (e.g., a 105mm tank round) or involved in a collision with a combat vehicle. A CCV is also destroyed by any artillery shell, mortar shell, or bomb that detonates within 25 meters. It may also be destroyed by a detonation occurring at greater distance, but with a probability that decreases linearly from 100% at 25 meters, to 0% at 50 meters (see Figure 2-2). When destroyed, a CCV will burn and smoke for 15 minutes, then remain on the battlefield as a blackened wreck.

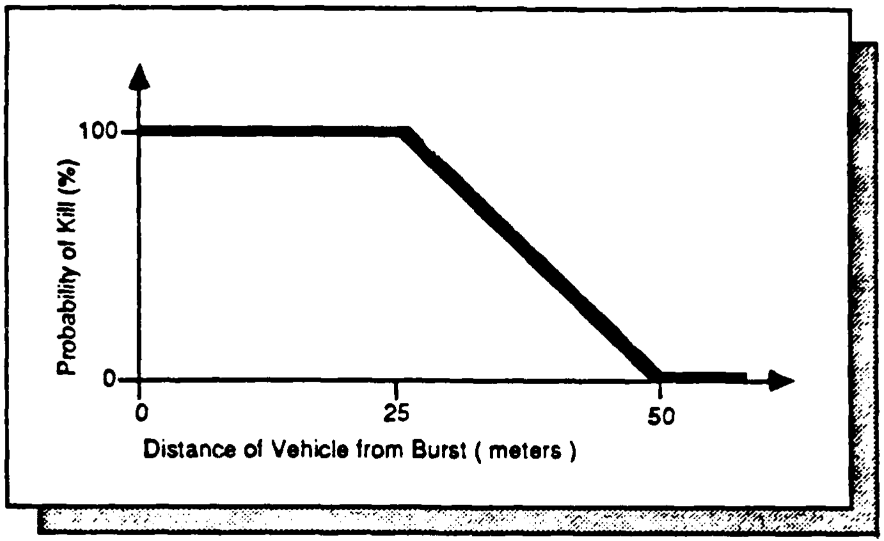


Figure 2-2. Probability of an indirect fire burst destroying a CCV.

All of the CCVs share these characteristics, but each also has its own peculiarities. Throughout the remainder of this chapter we will introduce the various CCVs and describe those characteristics that are unique to each.

2.3. Exercise Initialization

When the initial planning for an operation has been completed, the simulation for that operation can begin. Once the simulation has begun, the simulators themselves can be used to carry out further planning activities, such as reconnaissance.

A simulation begins with information entered at the SIMNET Control Console, usually by the battalion S3 or the BattleMaster. This information establishes the overall form of the exercise, and, once entered, it cannot be changed short of ending the simulation exercise and beginning a new one.

First, the battlefield terrain must be chosen from a short list of those terrain databases currently represented in all of the vehicle simulators and MCC systems participating in the exercise. This list of databases will contain from one to three choices. The choice of terrain determines the region within which all of the combat vehicles must operate. It also provides the MCC system with the information it needs to translate the UTM grid coordinates entered at the MCC consoles into locations on the battlefield.

Second, the organization of the unit participating in the exercise must be indicated to the MCC system. This information is referred to as the "task organization".

The MCC system must know which of four companies (A through D) are participating in the exercise, and on which side each company is playing. The two sides in a force-on-force exercise are somewhat arbitrarily labelled *offense* and *defense*, to distinguish them. A company may play on the side of offense or defense, or it may be specified as *mixed*, in which case the vehicles belonging to that company may be assigned individually to either side.

The MCC system must also be told which among several other elements of the battalion are participating. These optional elements include:

- the tactical operations center and the admin/log center. If they are participating, they can be positioned on the battlefield.
- the headquarters tank section and scout platoon. If they are participating, combat vehicles and crews can be assigned to these units.
- the *stealth jeep*. When this element is enabled (and when the stealth jeep simulator is developed), the BattleMaster will be able to place an invisible vehicle on the battlefield from which a senior commander may view the battle.
- the fire support element. If this element is enabled, the MCC system is able to simulate howitzers and mortars during the exercise.
- the air liaison officer. Enabling this element allows the MCC system to simulate close air support during the exercise.
- combat service support. With this element enabled, the MCC system can simulate ammunition trucks, fuel trucks, and maintenance teams.

Selecting one of these optional elements while beginning an exercise simply allows the function to be used during the exercise. However it does not mean that the function must be used, nor does it actually begin the simulation of the function. If the fire support element function is enabled, for example, then the MCC system will permit the fire support officer to subsequently start and use the artillery simulation during the exercise.

Once the terrain has been chosen and the unit's task organization has been described to the MCC system, the exercise is considered to have begun. At that point there are still no vehicles on the battlefield, but the SIMNET Control Console may then be used to initialize combat vehicle simulators, placing individual combat vehicles on the battlefield. The next section describes how this is done. The SIMNET Control Console may also be used to start the simulations of those optional elements (e.g., the fire support element) that were enabled for the exercise, resulting in computer controlled vehicles on the battlefield. Subsequent sections describe how each of the optional elements is initialized.

Combat vehicle simulators and optional elements may be initialized in any sequence, but each is normally initialized only once during the exercise. The BattleMaster may override this rule under special circumstances to re-initialize a simulator or element as described in Section 2.10.

2.4. Simulator Allocation and Combat Vehicle Deployment

Initializing combat vehicle simulators with the MCC system is a two-step process. The first step involves assigning the available simulators to the units participating in the exercise. Each unit, such as a company or scout platoon, can be allotted some number of simulators of each type. This allotment is done by the battalion S3 or BattleMaster, using the SIMNET Control Console. Of course, the MCC system will only allow simulators to be assigned to those units that, during exercise initialization, were indicated as participating in the exercise.

In the second step, the vehicles assigned to a unit are deployed on the battlefield. This is also done using the SIMNET Control Console, but typically each unit commander deploys his own vehicles. When deploying a vehicle, the commander specifies:

- the bumper number he wishes to assign to the vehicle. Each vehicle within his unit must have a unique, one- or two-digit bumper number.
- the location of the vehicle, entered as a six- or eight-digit UTM grid coordinate. This location must be within the bounds of the terrain database chosen for the exercise.
- the orientation of the vehicle relative to grid north. If the vehicle has a turret, the orientation of the turret is also specified.
- the quantities of fuel and ammunition of each type on board the vehicle, limited to the actual carrying capacities of the vehicle.
- if the commander's unit is a not a company described during exercise initialization as playing entirely on the offense force or entirely on the defense force, then the commander may specify the force to which the vehicle is attached.

This information is used to activate each combat vehicle simulator. Although each simulator can be activated in this way only once by the commander, not all simulators need be activated at once. The commander may, for example.

activate his own vehicle and use it to reconnoitre before completing his planning, returning to the SIMNET Control Console, and deploying the remaining vehicles in his unit.

2.5. Combat Service Support

The MCC system simulates fleets of fuel trucks, ammunition trucks, and maintenance teams for combat service support (CSS). These trucks are visible on the battlefield and vulnerable to combat damage like any of the MCC system's computer-controlled vehicles.

The ammunition trucks are M977 HEMTTs, each capable of carrying up to 22,000 lbs., or 705 cubic feet of ammunition. A truck may be loaded with any combination of six different kinds of ammunition, provided that neither the weight nor the volume capacity of the truck is exceeded. The six ammunition kinds are listed in Figure 2-3. Ten ammunition trucks are simulated.

AMMUNITION	UNIT	WEIGHT (pounds)	VOLUME (cubic feet)
105mm APDS-T	cartridge	68.49	2.25
105mm HEAT	cartridge	68.49	2.25
25mm APDS-T	can of 30 rds.	52.00	7.00
25mm HEI-T	can of 30 rds.	50.00	7.00
TOW	missile	87.00	4.50
DRAGON	missile	40.00	3.00

Figure 2-3. Ammunition weights and volumes.

The fuel trucks are M978 HEMTTs, each capable of carrying up to 2500 gallons of diesel fuel. Twelve of them are simulated.

The maintenance teams travel on the battlefield in M35A2, 2-1/2 ton trucks. Ten maintenance teams are simulated.

An additional three trucks are simulated for each of the battalion's companies. These M35A2, 2-1/2 ton trucks are designated as carriers of the company's prescribed load list (PLL), tools, and supplies. They are visible on the battlefield, but they neither travel nor serve any support function in the current implementation of the MCC system.

There are two supply depots to which the trucks must return for the supplies they will carry forward to replenish the combat vehicles. An ammunition truck may be loaded with ammunition when at a depot in the brigade support area called the ammunition distribution point. A fuel truck can take on fuel when at the class III distribution point in the brigade support area. These depots have no visible manifestation on the SIMNET battlefield; indeed they need not even be within the area covered by the terrain database. They are instead particular locations that anchor one end of the battalion's supply route, thus establishing the length of the supply route and, indirectly, the time required for supply trucks to traverse it.

Army doctrine prescribes *two* alternative ways of organizing an armor battalion's combat service support assets. When the CSS assets are organized as unit trains, they remain as a single fleet serving the entire battalion. When they are organized as echeloned trains, they are divided among combat trains that operate near the battle area and field trains that operate in the rear bringing supplies to the combat trains. The MCC system allows its CSS assets to be organized as either unit trains or echeloned trains.

2.5.1. Initialization

The simulation of combat service support is initialized from the SIMNET Control Console, typically by the battalion S4 (logistics) officer or some member of his staff. The S4 must specify how the CSS assets are to be organized, and the starting locations of the trains and depots. The individual trucks will then be given initial, default positions arrayed around those locations. If the trains are organized as unit trains, the S4 enters:

- the location of the battalion's support platoon. The ammunition and fuel trucks will then be given default positions surrounding that location, as shown in Figure 2–4.

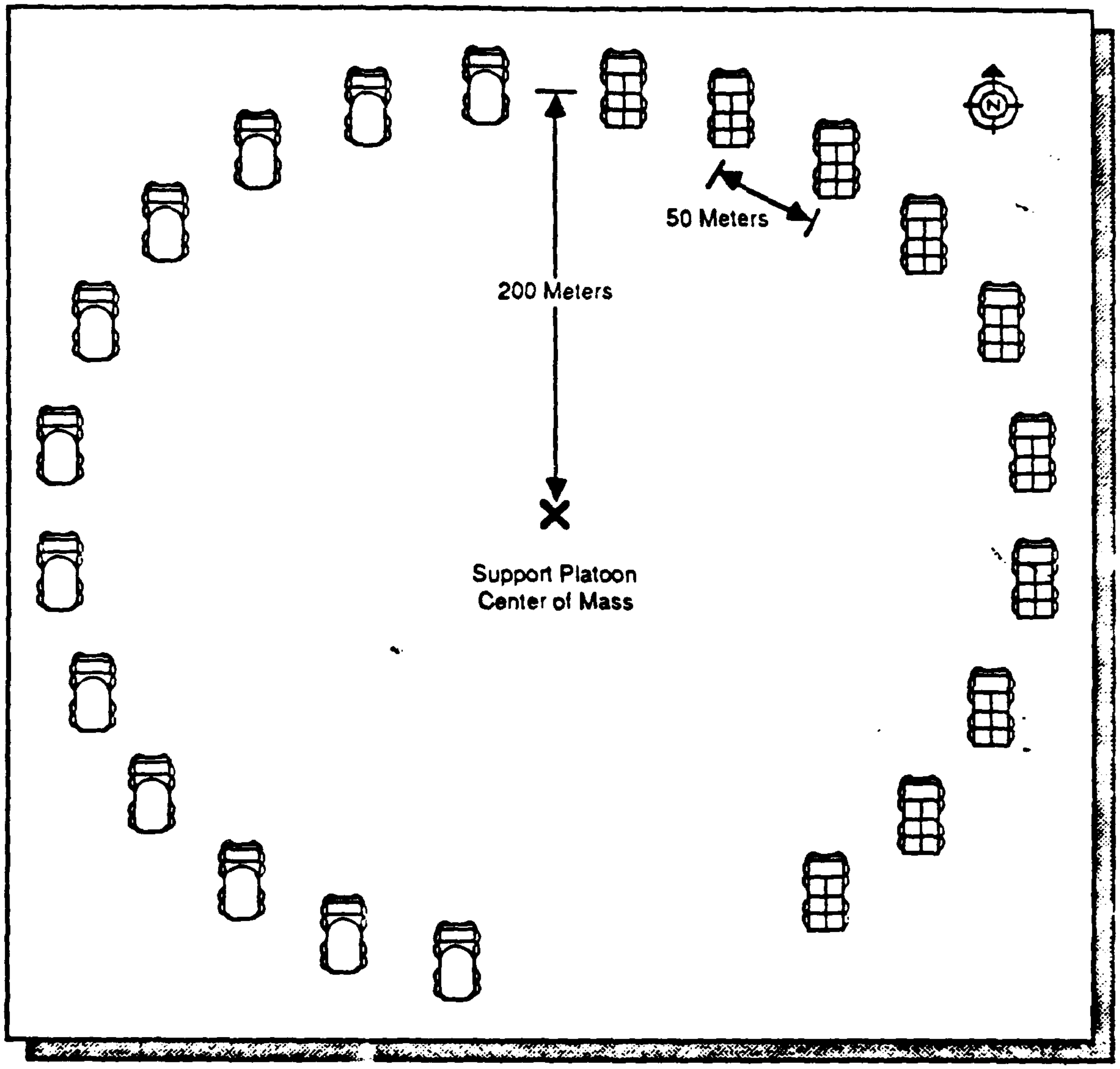


Figure 2–4. Default initial arrangement of the unit trains.

If the trains are echeloned, the S4 enters these locations:

- the location of the battalion's class III (fuel) supply point. The fuel trucks in the field trains will then be given default positions near that point, as shown in Figure 2–5.

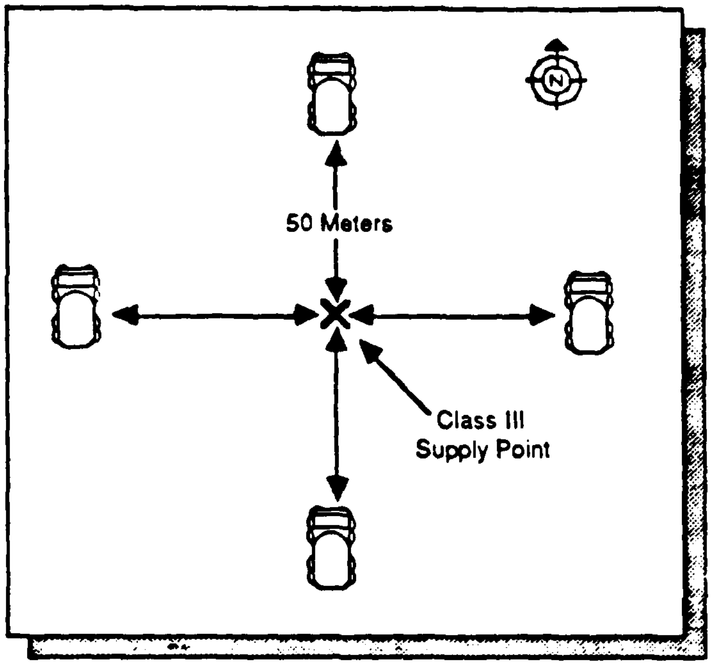


Figure 2-5. Default initial arrangement at the class III supply point.

- the location of the battalion's class V (ammunition) supply point. The ammunition trucks in the field trains will then be given default positions near that point, as shown in Figure 2-6.

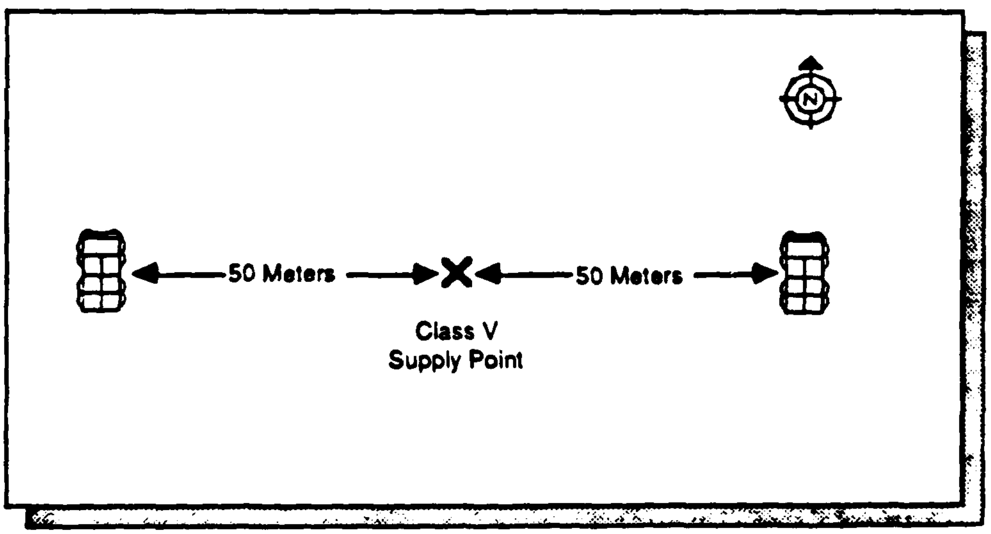


Figure 2-6. Default initial arrangement at the class V supply point.

- the location of each of the company trains. By default, each company's trains will be composed of two ammunition trucks, two fuel trucks, and three small supply trucks, positioned around the company's trains location as shown in Figure 2-7.

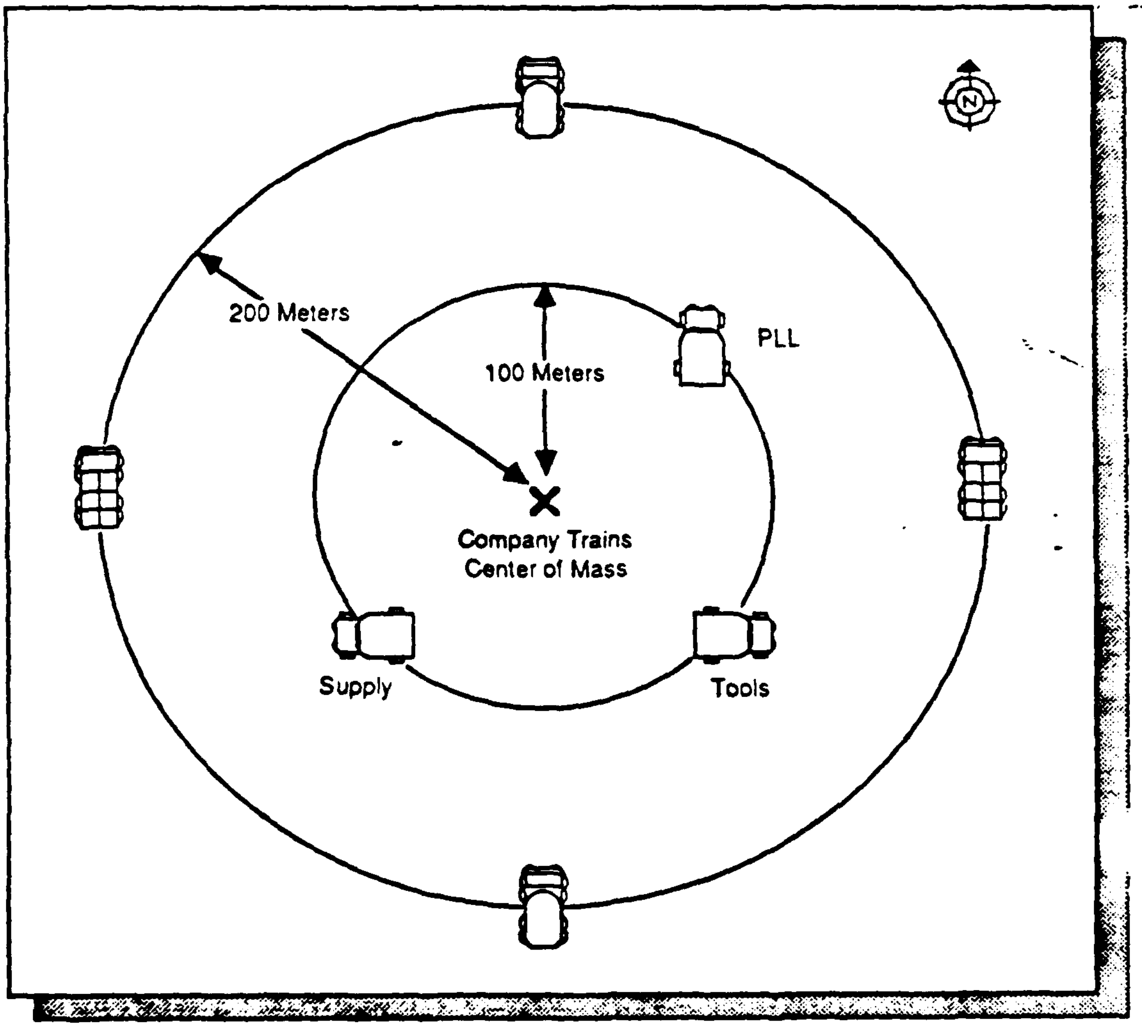


Figure 2-7. Default initial arrangement of a company's trains.

The S4 also enters the location of the battalion's unit maintenance collection point (UMCP). The maintenance teams will then be given default positions at the UMCP, as shown in Figure 2-8.

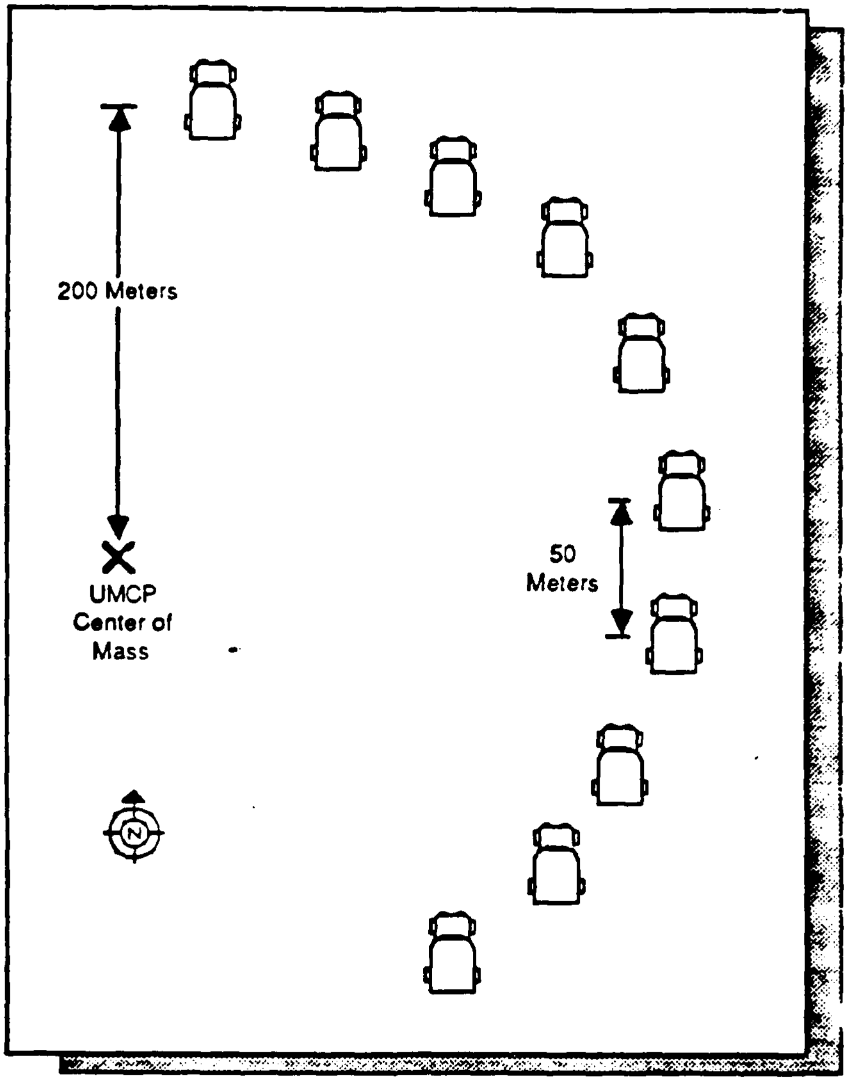


Figure 2-8. Default initial arrangement of the UMCP.

These locations establish a default position for each truck based on how the trains are organized. The S4 may override any of the positions, entering an explicit initial location for any truck. The supply trucks are also given default loads of fuel and ammunition by distributing the basic load for the battalion among them. The S4 can override those defaults, too, provided he doesn't exceed the weight or volume capacities of the trucks.

Each supply truck and maintenance team is assigned either to the battalion trains, or to one of the company trains. The MCC system will supply default company assignments for all trucks, but the S4 can override these defaults and assign an individual truck to any unit. This might be done, for example, when a platoon is cross-attached from one company/team to another, and some CSS assets must be cross-attached with it.

Each supply truck may also be assigned to the offense force or the defense force. A truck assigned to a company playing entirely offense or defense will itself be placed on the offense or defense force accordingly. If a truck is assigned to a company that was designated mixed during exercise initialization, or if it is assigned to the battalion, then the S4 can choose whether to place it on the offense force, the defense force, or neither. A truck that is on neither force appears friendly to players from both forces. It might be used by the S4 to resupply tanks on both sides in a force-on-force exercise.

2.5.2. Resupply

Once the simulation of combat service support has been initialized, the MCC system will make the Admin/Log and Maintenance Consoles available for use. The Admin/Log Console is used by the support platoon leader to coordinate resupply operations. This console provides a continuous display of the status of each supply truck, including its location and load composition. If a truck is in the process of travelling, its destination and estimated time of arrival are also displayed.

Not all trucks are available at all times. The MCC system will cause randomly selected trucks to break down at randomly selected times, regardless of whether they are stationary or travelling. This is done to mimic the availability of the actual trucks, and impose some uncertainty on supply operations. Breakdowns are limited so that at most two of the ammunition trucks and two of the fuel trucks will be disabled at any one time. A disabled truck becomes healthy again after a period of time that averages 15 minutes. While disabled, a truck can continue to transfer ammunition or fuel to combat vehicles, but it cannot move.

An ammunition or fuel truck that is not disabled, and has not already been destroyed in the exercise, can be dispatched to any location on the battlefield from the Admin/Log Console. When a truck is dispatched, the MCC system will compute and display an estimated arrival time for the truck based on the distance it must travel to reach its destination.

Like all CCVs, the truck is invisible while it is travelling. A special feature of an ammunition or fuel truck, however, is that it seeks out and approaches any nearby combat vehicle upon arriving at its destination. The truck will stop next to the combat vehicle nearest its original destination, provided there is one within 200 meters. If there is no nearby combat vehicle, the truck simply stops at its original destination. This affinity of trucks for combat vehicles gives the trucks a realistic ability to approach depleted or disabled combat vehicles.

A fuel truck can transfer fuel to a combat vehicle; an ammunition truck can transfer ammunition to a combat vehicle. In both cases, the truck and combat vehicle must be within 30 meters of each other, neither vehicle may have been destroyed, both vehicles must be stationary, the truck must have supplies on board to transfer, and the combat vehicle must have room for the supplies. The transfer may also be predicated on other conditions peculiar to the combat vehicle, such as a switch being in a certain position. It does not matter whether the truck and combat vehicle are on the same side in the exercise—combat vehicles can steal supplies from enemy trucks—nor does it matter whether the truck is disabled. As supplies are transferred, the information displayed on the Admin/Log Console is updated to reflect the decreasing load on board the truck. The rate at which the transfer takes place, and, in the case of a transfer of ammunition, the type of ammunition transferred, are both determined by the combat vehicle simulator.

The implementation of the MCC system described in this report permits only one combat vehicle to receive supplies from a truck at any one time. If a truck meets all of the necessary conditions for transferring supplies to two or more combat vehicles, one of those vehicles will be chosen arbitrarily by the MCC system to receive the supplies.

Ammunition and fuel trucks obtain their supplies from the appropriate depots in the brigade support area. The class III (fuel) supply point is modelled as having an infinite quantity of fuel; any fuel truck at that depot can always be reloaded to its maximum capacity. The class V (ammunition) supply point, however, is modeled as having a limited supply of ammunition. The current implementation of the MCC system fixes this supply at 1000 units of each type of ammunition at the beginning of the exercise. In a future version of the MCC system, we have planned to make the quantity of ammunition at the depot subject to the discretion of the BattleMaster.

The support platoon leader, working from the Admin/Log Console, controls the loading of supplies onto trucks at the depots. He can load an ammunition truck with any combination of different types of ammunition, subject to the availability of the supplies at the depot and the carrying capacity of the truck. He can also unload supplies from one truck at a depot in order to place those supplies on another truck. While a truck is away from the depot, however, its load only changes as a result of transferring supplies to a combat vehicle.

2.5.3. Repair and Recovery

The Maintenance Console is used by the battalion maintenance officer (BMO), and members of his staff, to coordinate repair and recovery of simulated combat vehicles.

To repair a combat vehicle, the BMO must first arrange a rendezvous between a maintenance team and the disabled vehicle. Maintenance teams are moved about on the battlefield in much the same way as ammunition and fuel supply trucks, but from the Maintenance Console rather than the Admin/Log Console. The Maintenance console provides a continuous display of the status of each maintenance team, including its location. If a team is in the process of travelling, its destination and estimated time of arrival are also displayed.

Like supply trucks, maintenance teams break down at randomly selected times for periods averaging 15 minutes, and while disabled they cannot be moved. But a maintenance team that is not disabled, and has not already been destroyed in the exercise, can be dispatched to any location on the battlefield. When a team is dispatched, the MCC system will compute and display an estimated time of arrival for the team based on the distance it must travel to reach its destination.

Maintenance teams are like other CCVs in that they are invisible while travelling. Futhermore, like supply trucks, maintenance teams will seek out and approach any nearby combat vehicles when they arrive at their destinations. A team will stop next to the nearest combat vehicle within 200 meters of its original destination. This gives the team a realistic ability to approach an immobile combat vehicle in order to repair it.

There is a correspondence between the repertoire of repairs of which the maintenance teams are capable, and the set of breakdowns modeled by the combat vehicle simulators. For example, a clogged fuel filter is one failure modeled by the M1 Abrams main battle tank simulator; replacing the fuel filter is a corresponding repair the simulated maintenance teams can perform. Figure 2-9 lists all of the repairs the MCC system simulates. These correspond to many of the failures currently modeled by the M1 simulator. (Not all combat vehicle failures can be repaired by the maintenance teams; some are simulated as being repaired by the vehicle crews themselves, and some others are irreparable.) Any maintenance team is capable of performing any of the repairs listed.

COMPONENT	ACTION	CATEGORY	DURATION (minutes)
Alternator	replace	electrical	30
Battery	replace	electrical	30
Engine Oil Filter	replace	automotive	15
Transmission Oil Filter	replace	automotive	15
Fuel Filter	replace	automotive	1
Starter/Pilot Relay	replace	electrical	30
Power Pack	replace	automotive	60
Service Brake	repair	automotive	15
Parking Brake	repair	automotive	15
Turret Traverse Drive	repair	turret	30
Turret–Mount Interface	repair	turret	240
Gun Elevation Drive	repair	turret	30
Stabilization System	repair	turret	30
Laser Rangefinder	repair	turret	20
Fuel Pump	repair	automotive	30

Figure 2-9. Combat vehicle repairs simulated by the MCC system.

The maintenance teams require a certain amount of time to perform each repair. Replacing a fuel filter always takes 1 minute, for example, and repairing a turret-mount interface always takes 4 hours. These times are simple approximations. Their only purpose is to ensure that breakdowns constrain the maneuver force, requiring it to coordinate its maintenance operations carefully in order to remain effective.

A maintenance team can repair a combat vehicle as long as that team's truck and the combat vehicle are within 30 meters of each other, neither vehicle has been destroyed, and both vehicles are stationary. These conditions must remain in effect throughout the duration of the repair.

A single maintenance team can simultaneously perform more than one repair, provided that no two of the repairs being performed draw on the same category of maintenance skills: automotive, electrical, or turret. For example, a maintenance team can, in 30 minutes, repair both the stabilization system and the fuel pump of a combat vehicle, but not the stabilization system and the gun elevation drive. The stabilization system and gun elevation drive must be repaired at separate times, requiring 60 minutes altogether.

The Maintenance Console is used by the BMO to start a repair involving a particular maintenance team and combat vehicle. In starting a repair, the BMO specifies the repair to be performed, and the console responds by predicting the time at which the repair will be completed. The BMO's choice of what to repair will be based on a diagnosis of the symptoms, such as engine warning lights, that are observed in the combat vehicle simulator. These symptoms may be reported over the radio by the vehicle's crew, or directly observed by the battalion's maintenance personnel.

The Maintenance Console displays all repairs underway and completed, along with information such as which vehicles are being repaired, what is being repaired on them, and when the repairs are due to be completed. The BMO can cancel any repair in progress. He can also interrupt a repair by dispatching the maintenance team performing it to a new location on the battlefield. When a repair to a combat vehicle is thus aborted, the combat vehicle is left unaltered, as if no repair had been started on it.

When the time arrives for a repair to be completed, the MCC system notifies the combat vehicle simulator of the repair. That simulator updates the state of its vehicle model to reflect any changes the repair may cause. If the repair was not appropriately chosen so as to correct some simulated deficiency in the vehicle, the vehicle's crew will see no improvement as a result of the repair.

The Maintenance Console may also be used to direct the recovery (towing) of disabled combat vehicles to locations, such as the UMCP, where they can be repaired in greater safety. The current *implementation of the* MCC system does not simulate recovery vehicles like the M88, but the maintenance teams are modeled as being capable of performing recovery.

Once a maintenance team has rendezvoused with a combat vehicle, the BMO can use that team to recover the vehicle. At the Maintenance Console he orders the vehicle recovered and enters the location to which it should be towed. Following a five minute delay—representing the time required to hitch up the disabled vehicle—both the maintenance team's truck and the combat vehicle disappear from the battlefield. The combat vehicle simulator is also deactivated by the MCC system, making that vehicle inoperative and its display screens blank. Towing is simulated like all CCV travel: at 25 kilometers per hour, in a straight line to the destination. When towing is complete, both the maintenance team and the combat vehicle appear on the battlefield at the towing destination. The combat vehicle simulator is re-activated by the MCC system, making it operative again with precisely the same failures and supplies it had when deactivated. Its display screens will show the battlefield around its new location.

2.6. Fire Support

The MCC system simulates two sets of indirect fire weapons for use by the battalion's fire support officer (FSO): a battery of eight, 155mm self-propelled howitzers, type M109; and a platoon of six, 107mm mortar carriers, type M329.

2.6.1. Howitzer Battery

The howitzer battery is modelled as being in a role of direct support to the armor battalion. This means that, although the fire support officer can perform fire missions with the battery, he does not oversee the movement or resupply of the battery. In a future version of the MCC system, the BattleMaster will be able to assume the responsibilities of the battery commander to move and resupply the battery's ammunition.

The eight howitzers of the battery are divided into two platoons of four; each platoon can be assigned a fire mission independently of the other, or both platoons can be given the same mission.

A simulated howitzer will fire a series of rounds at a rate of 4 rounds per minute for the first 3 minutes, and 1 round per minute thereafter.

The range of the simulated howitzer is from 1000 to 18,300 meters, with no restriction as to its azimuth of fire. The shells modeled are high explosive, type M483, with a choice of two types of fuze: point detonating, or variable-time set to a 20-meter height-of-burst. Figure 2-10 shows the time of flight of rounds fired various distances, as derived from *Firing Table FT 155-AN-1*[4].

RANGE (kilometers)	CHARGE	TIME OF FLIGHT (seconds)
1	3W	3.5
2	3W	7.2
3	4W	9.9
4	4W	13.6
5	4W	17.7
6	5W	20.0
7	5W	24.3
8	5W	29.2
9	6W	29.5
10	6W	34.4
11	7W	40.5
12	7W	38.1
13	7W	43.7
14	7W	51.3
15	8	45.2
16	8	50.9
17	8	58.5
18	8	68.0

Figure 2-10. Time of flight of howitzer rounds.

Impacts of rounds other than adjustment rounds are scattered in both the north–south and east–west directions by random amounts uniformly distributed over the range -30 to +30 meters.

Figure 2-11 shows how the battery is typically deployed on the ground. The actual arrangement will vary with the terrain since, like all CCVs, each gun may be moved slightly by the MCC system so as to be on open, navigable ground.

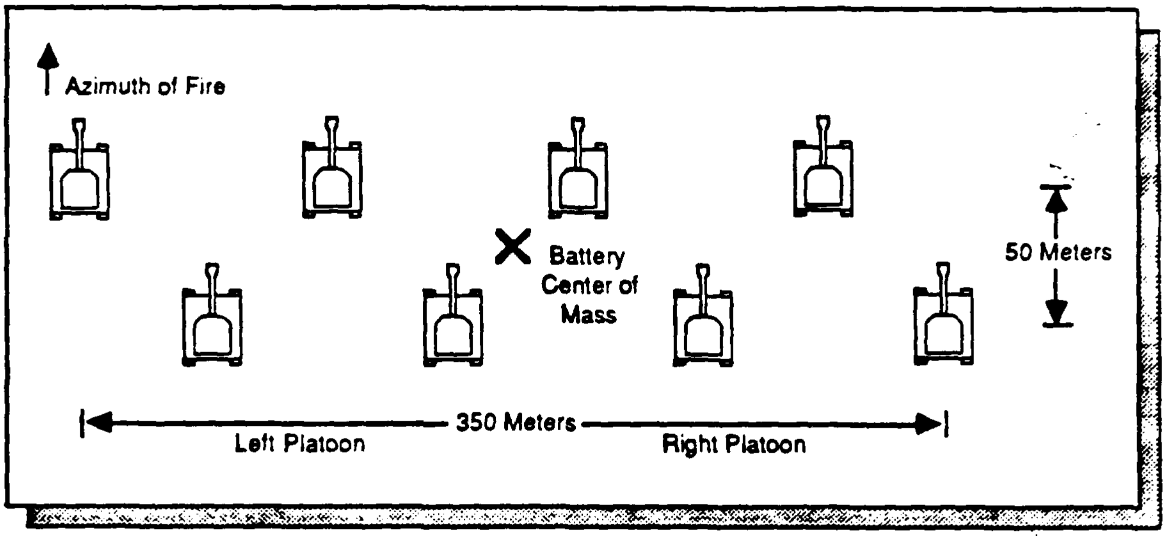


Figure 2–11. Deployment of the howitzer battery.

2.6.2. Mortar Platoon

The mortar platoon modeled is that which is organic to the maneuver battalion. The fire support officer is fully responsible for this platoon. He can not only perform fire missions with it, but also move it about on the battlefield during the exercise. In a future version of the MCC system, he will also be responsible for resupplying the mortar platoon with ammunition.

The six mortars of the platoon are divided into two sections of three; each section can be assigned a fire mission independently of the other, or both sections can be given the same mission.

A simulated mortar will fire a series of rounds at a rate of 20 rounds during the first minute, and 10 rounds during each subsequent minute.

The range of the mortar is from 1000 to 5600 meters, with no restriction as to its azimuth of fire. The shells modeled are high explosive, type M329, with a choice of two types of fuze: point detonating, or proximity set to a 20–meter height–of–burst. Figure 2–12 shows the time of flight of rounds fired various distances, as derived from *Firing Table FT 4.2–H–2*[5].

RANGE (kilometers)	ELEVATION (mils)	TIME OF FLIGHT (seconds)
1.0	1065	19.3
1.5	800	18.1
2.0	800	21.1
2.5	800	23.8
3.0	800	26.3
3.5	800	28.7
4.0	800	30.8
4.5	800	32.7
5.0	800	34.8
5.5	800	36.8

Figure 2-12. Time of flight of mortar rounds.

Impacts of rounds other than adjustment rounds are scattered in both the north-south and east-west directions by random amounts uniformly distributed over the range -30 to +30 meters.

Figure 2-13 shows how the platoon is typically deployed on the ground. The actual arrangement will vary for the same reasons mentioned earlier as applying to the howitzer battery.

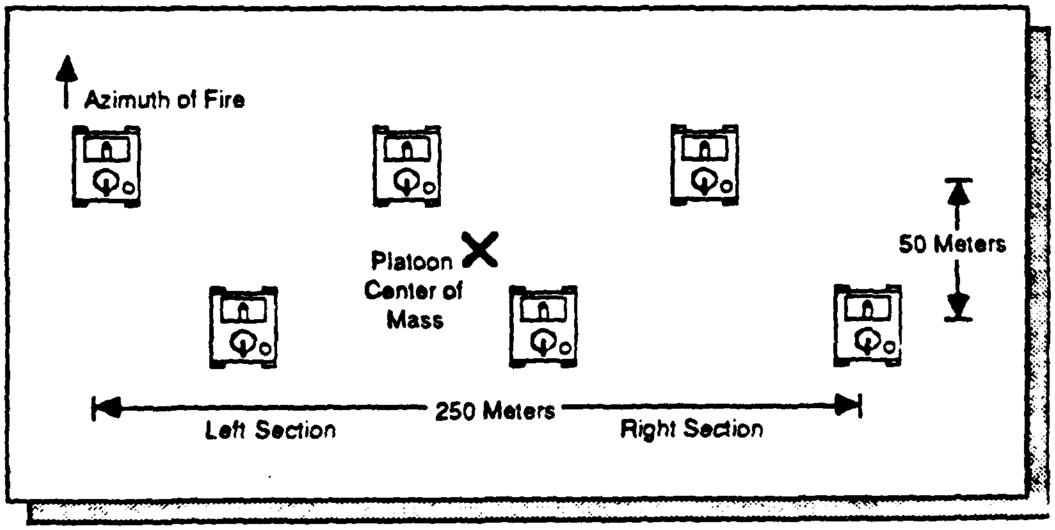


Figure 2-13. Deployment of the mortar battery.

2.6.3. Initialization

The Fire Support Element simulation is initialized by the fire support officer from the SIMNET Control Console, and, thereafter, directed by him from the Fire Support Console. In initializing the simulation, the FSO specifies—for both the howitzers and the mortars—where the guns are to be located and at what mounting azimuth. He also specifies how many rounds of each type of ammunition they have on hand. Once this information has been entered, the Fire Support Console is activated.

2.6.4. Fire Missions

Working from the Fire Support Console, the FSO can give three different types of missions to the howitzers and mortars: adjustments, fires for effect, and final protective fires. An adjustment results in a single round being fired at the target by a single gun from among those chosen for the mission. When firing for effect, all guns fire a specified number of rounds in a parallel sheaf that results in a pattern of bursts on the ground mirroring the positions of the guns in the battery. A final protective fire produces a continuous curtain of fire stretching between two points on the terrain. The impacts from the first volley of a fire for effect or final protective fire occur in unison; on subsequent volleys the impacts occur at scattered times.

Howitzers and mortar carriers are visible on the battlefield like any other CCVs. Moreover, as they fire, their muzzle flashes can be seen and heard from nearby combat vehicles. Once a gun has been destroyed in the exercise, of course, it is no longer capable of firing.

The target of an adjustment or fire for effect mission may be specified in any of three ways. If the UTM grid coordinate of the target is known, it may be used. A polar plot may also be used, in which case the location of an observer and the heading and distance from that observer to the target are entered. Finally, the target may be specified as a shift from some point, the coordinate of which is known.

After firing an adjustment or fire for effect, the FSO can fire another adjustment or fire for effect against the same target with the same or a different set of guns. He can also shift the target location some meters before firing additional rounds. A typical engagement begins with the firing of one or more adjustment rounds to locate the target with the help of an observer on the battlefield who can see the rounds impacting. The observer notifies the FSO by radio of how far and in what direction the adjustment rounds are missing the target so that the FSO can shift the location at which the adjusting gun is firing. When an adjustment round falls sufficiently close to the target, the observer may then request a fire for effect, which results in the FSO firing as many rounds as necessary to destroy or neutralize the target.

It is normal practice for an FSO to prepare a list of likely indirect fire targets prior to a real operation. This target list is then distributed to the batteries so they can precompute the parameters they will need in order to engage those targets when called upon to do so; and it is distributed to the artillery observers and combat units so they can call for fire on those targets. The FSO practices this same procedure for a SIMNET operation, except that the target list is entered into the Fire Support Console instead of being given to the batteries. Each target on the list has associated with it a UTM grid coordinate and a unique target number. Once a target has been included in the target list, the FSO need only specify its target number in order to fire upon it. Those targets located through a series of adjustments can be readily added to the list as well.

The Fire Support Console maintains a schedule of fire missions to aid the FSO in coordinating fire support for a maneuver. With this schedule, the FSO can preplan fire for effect missions and have them performed automatically at specified times. He might, for example, plan an artillery concentration to be fired at the start of an operation. By entering that mission into the schedule the FSO can request the mission ahead of time and have it performed as scheduled. When the time comes to perform a scheduled mission, the mission will simply be dropped, however, if the guns assigned to perform it are out of ammunition, beyond range of the target, or inoperative.

2.6.5. Displacement

The fire support officer can move the mortars about on the battlefield from the Fire Support Console. Each section can be moved independently, at different times or to different destinations. The mortars travel, like all CCVs, at 25 kilometers per hour. They cannot fire while travelling, nor can they fire while "setting up", during the first 5 minutes after they have reached their destination.

2.7. Close Air Support

The MCC system provides a limited simulation of close air support (CAS), appropriate for the training of combat unit commanders in the use and consequences of this form of combat support. The simulation considers a sortie to be a single A10 carrying twelve, 500 pound anti-armor cluster bomb units. Although the aircraft itself is neither visible from the battlefield nor individually simulated by the MCC system, the effects of its attack—the bombs it releases—are readily apparent to combat vehicle crewmembers.

2.7.1. Sortie Limits

Two types of close air support missions are distinguished by the MCC system: preplanned missions, and on-call missions. A preplanned mission is one that, as its name implies, is anticipated and scheduled well in advance of the mission. An on–call mission, on the other hand, involves requesting aircraft that are on alert to attack as soon as possible.

In allowing the construction of a realistic exercise scenario, the MCC system allows limits to be placed on the total number of sorties, for both kinds of missions, available during the exercise. It also allows limits to be placed on the number of sorties, of those totals, that may be preplanned. New limits must be entered each new day of a multi–day exercise, but any sorties uncommitted at the end of a day may not be carried over to the following day. These restrictions are designed to allow the imposition of realistic constraints on the availability of close air support, comparable to the constraints that may be expected in a real operation.

The simulation of close air support is performed under the direction of the battalion's air liaison officer (ALO). He begins the simulation, at the SIMNET Control Console, by specifying how many sorties—both total sorties and preplanned sorties—are to be available for the first day of the exercise. Once this has been done, the Close Air Support Console becomes active and the ALO uses that console for the remainder of the exercise.

At any time during the exercise, the BattleMaster can use the SIMNET Control Console to make additional sorties available for close air support. He would do this at the beginning of each day, entering the number of sorties allotted by the imaginary air force for use by the battalion during that day. He may also do this during the day, increasing the limits on the number of sorties available that day.

2.7.2. Scheduling Missions

From the Close Air Support Console, the ALO may plan the use of close air support, call up air strikes against battlefield targets, retain aircraft on station (i.e., over the battlefield), redirect sorties in progress to alternate targets in response to changing battlefield conditions, and record the effects of completed missions.

Visible on the screen at all times is a table displaying the schedule of CAS missions in various stages of completion: past, current, and future missions. The schedule includes, for each mission:

- the location of the target the mission is to attack;
- a description of the target as reported by an observer on the battlefield;
- the time at which the target is to be attacked;
- the number of sorties participating in the attack;
- the direction the aircraft are to travel over the target (i.e., the "run–in heading"); and

- for past missions, the results of the mission as reported by an observer.

The ALO may add to this schedule both preplanned and on-call missions to be performed during the current day. When adding a preplanned mission, he specifies when the aircraft are to be over the target; this must be at least 25 minutes in the future because the aircraft are expected to requires some time to reach their target. When he requests an on-call mission, it is scheduled for 25 minutes hence. The period 25 minutes is intended to represent an approximate, average time required for close air support aircraft to reach their targets during a real operation.

2.7.3. Completing Missions

Missions may be cancelled or changed any time before the aircraft have released their payloads. The ALO may, for example, redirect the aircraft to a new target location because the situation on the battlefield has changed since the mission was scheduled. Also, once the aircraft have reached their target, the ALO may have them loiter over the target area (i.e., remain "on station") for a period of up to 10 minutes. He may, for example, do this while checking by radio with his observer on the battlefield, before having the aircraft bomb the target. The simulated aircraft cannot remain on station longer than 10 minutes without finding that they are low on fuel ("bingo fuel"), at which point the ALO must send them home either with or without their payloads.

While the sorties that constitute a mission are on station, the ALO may command them to bomb their target ("clear hot"). The simulated attack will occur immediately. The aircraft pass over the battlefield in the vicinity of the target at intervals of 5 seconds, each dropping their entire load of 12 bombs. The bombs deploy in the patten shown by Figure 2-14.

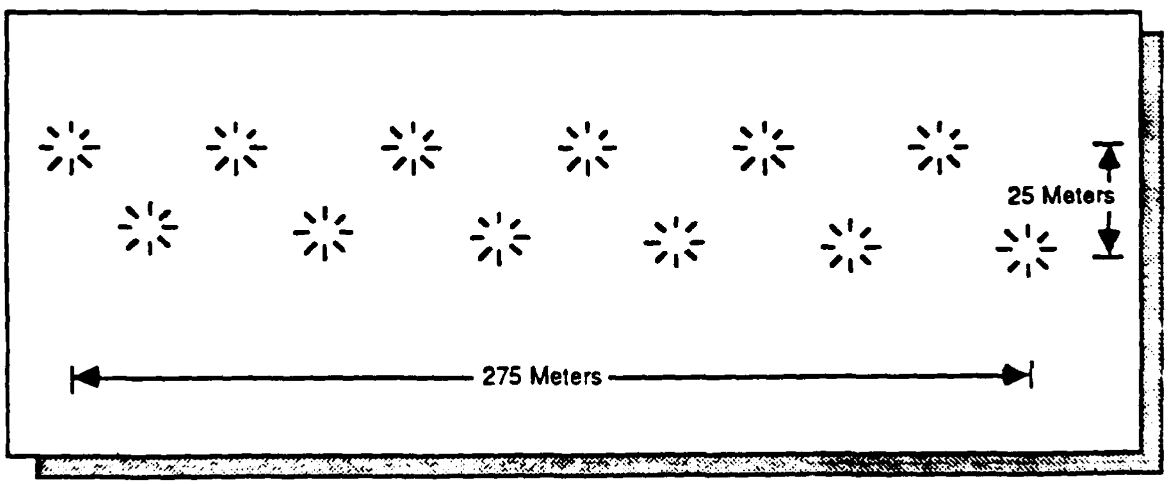


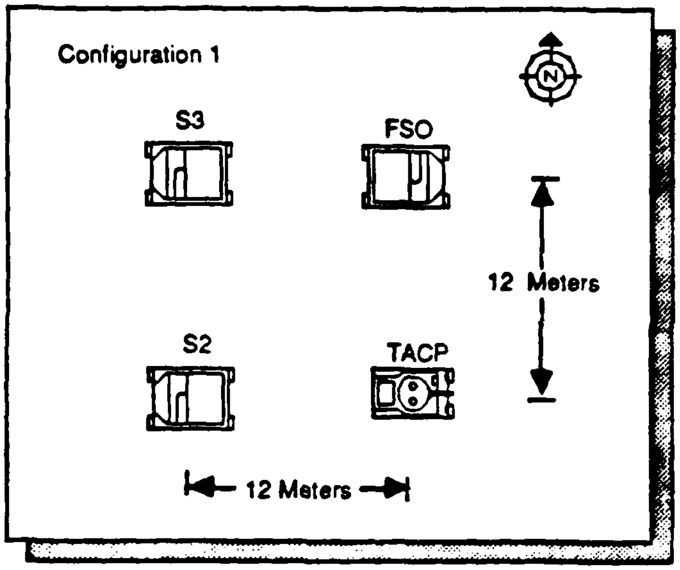
Figure 2-14. Pattern of bomb explosions.

2.8. Command Posts

The MCC system includes a battalion tactical operations center and admin/log center that have both real and simulated components. The real command posts are physical places from which the battalion officers can work while staying in contact by voice radio with soldiers in their vehicle simulators. These real command posts have manifestations on the battlefield as command post CCVs simulated by the MCC system.

2.8.1. Tactical Operations Center

The tactical operations center (TOC), which usually serves as the headquarters for operational control of the battalion, appears on the battlefield as a collection of three M577 command post vehicles and one M3 Bradley fighting vehicle. These vehicles may be arranged in either of the two configurations shown in Figure 2-15. The choice of a TOC location and its arrangement are specified at the SIMNET Control Console, any time after exercise initialization.



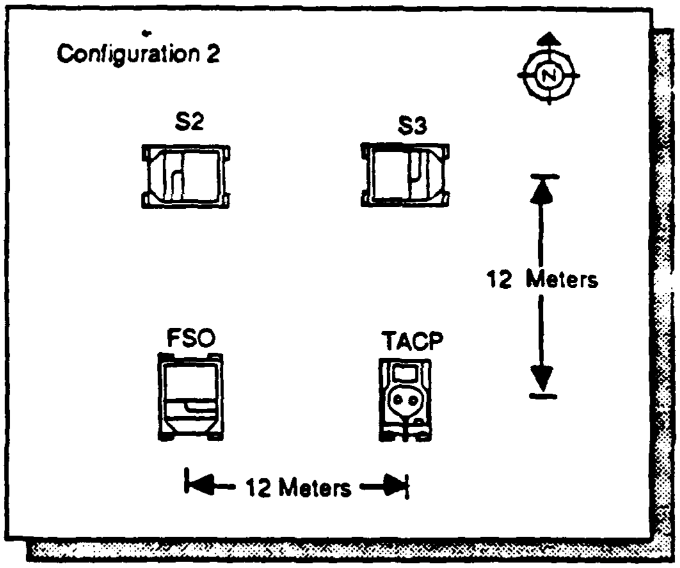


Figure 2-15. Alternate arrangements of the tactical operations center.

The physical TOC contains space for the S2, S3, FSO and ALO, who operate from the TOC as they would during a real operation. The TOC includes voice radios for each of these individuals, allowing them to communicate on the various FM radio nets of the battalion. The consoles controlling fire support and close air support are also present in the TOC, where they can be used by the the FSO and ALO.

2.8.2. Admin/Log Center

The admin/log center serves as the headquarters for the battalion's combat service support organization. It appears as a single M577 command post vehicle on the battlefield. The location of this vehicle is specified at the SIMNET Control Console, any time after exercise initialization.

The physical admin/log center contains space for the S1, S4, support platoon leader, and BMO, who operate from the admin/log center as they would during a real operation. The admin/log center includes voice radios for these individuals, allowing them to communicate on the various FM radio nets of the battalion. The consoles controlling admin/log and maintenance are also present in the admin/log center, where they can be used by the the support platoon leader and BMO.

2.8.3. Voice Radios

The MCC system includes a hardware interface for controlling power to seven of the voice radios present in the TOC and admin/log center. All of these radios are powered off when the exercise begins and ends. Individual radios are powered on, and thus made available for use, under various conditions during the exercise. These are the radios and the conditions under which they are enabled and disabled:

- A radio for the battalion S2 is enabled when the TOC is placed on the terrain or reconstituted. The radio is disabled when the S2's M577 vehicle within the TOC is destroyed.
- A radio for the battalion S3 is enabled when the TOC is placed on the terrain or reconstituted. The radio is disabled when the S3's M577 vehicle within the TOC is destroyed.
- A radio for the fire support officer is enabled when the fire support element is initialized. It remains on for the remainder of the exercise.
- A radio for the air liaison officer is enabled when close air support is initialized. It remains on for the remainder of the exercise.
- An admin/log center radio is enabled when the admin/log center is placed on the terrain or reconstituted. The radio is disabled when the admin/log center is destroyed.
- Radios for the support platoon leader and battalion maintenance officer are enabled when combat service support is initialized. They remain on for the remainder of the exercise.

2.9. Static Vehicles

The SIMNET Control Console can be used by the BattleMaster to create and place arbitrary instances of CCVs on the battlefield. These CCVs are called *static vehicles*. There is no particular role for static vehicles in a normal training exercise, but they are useful for demonstrating the SIMNET system, and teaching combat vehicle crews how to recognize the various types of vehicles present on the simulated battlefield.

At any time during an exercise the BattleMaster may create new static vehicles, modify the appearances of existing ones, remove static vehicles, or view their status. Up to thirty static vehicles can exist at once.

Each static vehicle may have a name assigned by the BattleMaster for identification. Also, each static vehicle has a specified type, or appearance, which may be any of the various kinds of vehicles modeled by SIMNET. The current implementation permits static vehicles to be any of: M1, M2/3, Soviet T72, Soviet BMP, ammunition truck, fuel truck, 2-1/2 ton truck, recovery vehicle, self-propelled howitzer, or mortar carrier. For those vehicles, such as trucks, which appear on the battlefield with either of two colorations—friendly or enemy—the BattleMaster chooses whether the vehicle is to appear as friendly to all observers, or as enemy. He also specifies an orientation (azimuth) and a location on the battlefield for the static vehicle. The BattleMaster may change any of the parameters of a static vehicle as often as desired.

When a static vehicle is destroyed by direct fire, indirect fire, or a collision, it responds—as does any CCV—by catching fire. From the SIMNET Control Console, the BattleMaster can see which static vehicles have been destroyed. He can also reset individual static vehicles, or all at once, to make them appear healthy once more.

The MCC system's simulation of some of the static vehicles includes a small amount of animation: the turrets and gun tubes of those vehicles that have them can be seen on the battlefield to scan slowly back and forth.

2.10. Reconstitution

The SIMNET Control Console gives the BattleMaster the special power of being able to restore to full operation any simulated element, be it a crewed combat vehicle or a computer-controlled vehicle. This is called *reconstitution*. It is a feature intended to be used not during normal, training exercises, but only in extraordinary circumstances, such as while demonstrating or testing the SIMNET system.

The following is a list of things that may be reconstituted.

- Any combat vehicle simulator can be redeployed, as if for the first time, at initialization. Any of the vehicle's parameters, such as its location or quantity of supplies, may be changed while it is being reconstituted. If the vehicle is damaged, suffering some failure, or even destroyed, it will be returned to perfect health.

- Any ammunition truck, fuel truck, or maintenance team can be redeployed with a new location, company assignment, and, if applicable, fuel or ammunition load. If the vehicle is destroyed, it is made healthy again.

- The TOC and the admin/log center can be redeployed to a new or the same location. Any destroyed vehicles are made healthy, and any voice radios associated with those vehicles are enabled once more.

- The howitzer battery or mortar platoon can be redeployed, perhaps with a new location, azimuth of fire, or supply of ammunition. Any of the individual howitzers or mortars within the unit that are destroyed, are made healthy again.

Of course, a vehicle can only be reconstituted if it has already been initialized at some point during the exercise.

2.11. Ending the Exercise

Using his discretion, the BattleMaster may at any time end the exercise by issuing a command at the SIMNET Control Console. When he does so, all simulators and MCC consoles are reset, and the SIMNET system is made ready to perform a new simulation exercise.

2.12. Statistics Compiled by the MCC System

The MCC system compiles and records statistics that summarize the simulation exercises for which it is used. For each combat vehicle simulator involved in an exercise, the MCC system will record the following information about the use of that simulator's vehicle:

- the total distance travelled by the vehicle;

- the amount of fuel consumed by the vehicle; and

- the number of rounds of each type of ammunition fired by the vehicle.

For the overall exercise, the MCC system will record:

- the starting date and time, and duration of the exercise;

- the total distance travelled by all combat vehicles participating in the exercise;

- the total quantity of fuel consumed by all combat vehicles;
- the total number of rounds of each type of ammunition fired by all combat vehicles;
- the number of howitzer and mortar shells of each fuse type fired during the exercise; and
- the number of bombs dropped by close air support aircraft during the exercise.

At the close of each exercise, a record of this information is appended to a file residing at the MCC host.

2.13. Host Terminal

The MCC system includes a terminal called the *host terminal*, which is directly attached to the MCC host. Through this device a SIMNET technician may interact with the MCC system, while it is running, to diagnose and overcome certain problems that may arise. The following subsections describe the services available to a technician who is using this terminal.

2.13.1. Status

In response to a command entered at the host terminal, the MCC system will display a "snapshot" summary of its internal state. This summary includes the status of each software module and each major data structure within the MCC host, and the status of each combat vehicle simulator. The information presented may be used to diagnose a problem, such as the failure of a console to respond to its user. The status summary is described more thoroughly in Chapter 13 of this document.

2.13.2. Restart

The MCC system may be ordered, via the host terminal, to restart the hardware and software associated with the Admin/Log, Maintenance, Fire Support, or Close Air Support Console. This might be done, for example, to recover from an inadvertent interruption to the power supplying one of those consoles. When a console is restarted, some of the information entered into the console up to that point will be lost. When the Fire Support Console is restarted, for example, the target list is lost, but the locations of the howitzers are retained. Future versions of the MCC system will be able to retain more information during console restarts.

2.13.3. Date and Time

Each MCC console provides a continuous display of the current date and time, as known to the MCC system. The MCC system must have the correct time because many of the requests it receives—such as requests for close air support—include some specification of a time. If the MCC system has the wrong date or time, it can be corrected during an exercise by entering a command at the host terminal. The time displayed by the MCC consoles will then be updated to the new, correct time entered.

3. ARCHITECTURE

In this chapter we discuss the architecture of the MCC system. We use the term architecture here to mean the way in which a computer system is partitioned into software and hardware components. The hardware components of the MCC system included in this architecture are a network, several computers, and their peripherals. The software components are modules of software that run on the various computers.

The architecture of the MCC system is derived partly from the functional requirements surveyed in the previous chapter. The following goals have also influenced this architecture:

- the MCC system must have user interfaces that are highly interactive, that are visual, and that provide the user with prompt response;
- the MCC system must have sufficient performance capabilities to support a maneuver battalion of seventy combat vehicles in a simulated battlefield environment involving hundreds of vehicles altogether; and
- the architecture must support the evolutionary development of the MCC system through a continuing process of "rapid prototyping"

In describing the MCC system, we will show how its architecture is derived from these goals. This chapter introduces first the hardware components of the MCC system, and then the software components occupying that hardware. In introducing these components, we will go no further here than to sketch the essential features of their operation. Subsequent chapters describe the software components in greater detail.

3.1. Hardware Overview

The MCC system is a distributed system composed of several, physically separate computers that are interconnected by a local area network. Figure 3–1 shows the hardware components and how they are interconnected.

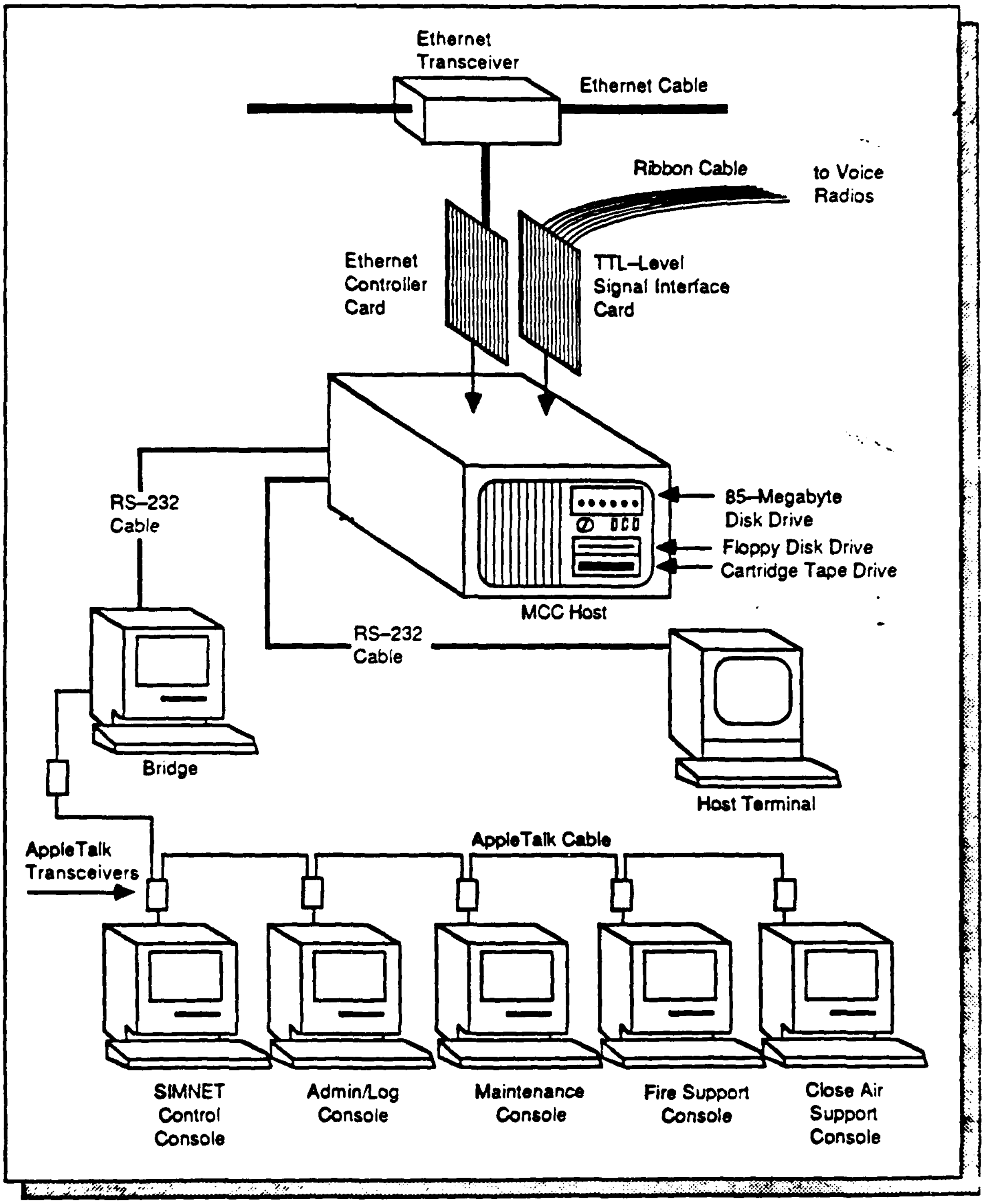


Figure 3-1. MCC system hardware.

No custom–built hardware is incorporated in the MCC system. All components are "off the shelf" products readily available from commercial sources.

3.2. Console Hardware

Each of the consoles serving a user of the MCC system is implemented by a dedicated Macintosh computer. Because a single one of these Macintosh computers fully implements the user interface of a single console, the MCC system is able to provide its users with interfaces that are highly interactive and responsive. Futhermore, since most of the interactions between a user and his console are handled locally by the dedicated console Macintosh, the response provided that user is largely unaffected by the overall load on the MCC system.

The graphics capabilities of the Macintosh make it an appropriate choice as an MCC console. In addition to a bit–mapped screen and a mouse, the Macintosh has built–in software that eases the development of graphical user interfaces. This software, called the Toolbox, implements visual devices such as icons, windows, and menus. Programs written for the Macintosh can use the Toolbox to support user interfaces that are highly interactive.

Each Macintosh computer contains a Motorola MC68000 microprocessor capable of executing about one half million instructions per second. An operating system and the Toolbox software reside in ROM within the computer. The Macintoshes used as MCC consoles have 512K or 1M bytes of RAM, a 512 by 342 pixel monochrome display, and a floppy disk drive. There is an interface built into each Macintosh that supports its attachment to a local area network called an AppleTalk network.

3.3. Host Hardware

The MCC system must participate on a high–speed local area network in order to communicate with other components of the SIMNET system, such as the combat vehicle simulators. It must also consult large volumes of terrain data, and perform detailed, arithmetic calculations. For these functions, the MCC system includes a host computer which is somewhat more powerful than the Macintoshes used as consoles.

A Masscomp 5600 system was selected to be the MCC host computer for the following reasons:

- The system runs a version of UNIX® enhanced to support real–time applications. The enhancements include a provision for fixing the priority of a process so that it may intentionally monopolize a processor and thereby provide prompt response to real–time events. This version of UNIX also supports a variety of

® UNIX is a registered trademark of AT&T Bell Laboratories.

interprocess communication mechanisms, described below in section 3.7.2.

- The system has the hardware features needed by an MCC host, such as floating point hardware and sufficient disk storage to hold terrain databases. It also has an industry standard bus allowing it to support an interface to the high-speed SIMNET local area network.

- The processing power of the system can be increased, if needed, by adding a second processor, additional memory, and more powerful floating point arithmetic hardware.

- The same type of system also serves as host to the combat vehicle simulator software. The choice of a common system means certain software can be shared, and fewer different types of spare parts must be stocked at SIMNET sites.

The Masscomp 5600 system is based on a Motorola MC68020 microprocessor capable of executing about two-and-one-half million instructions per second. The chassis of this system contains a Multibus backplane with space for several I/O interface cards. When configured to be an MCC host, the system includes a Motorola MC68881 floating point co-processor for arithmetic calculations, two megabytes of main memory, a disk drive with a capacity of 85 megabytes or more, a floppy disk drive, a cartridge tape drive, an eight-channel RS-232 interface, an Ethernet local area network interface, and a TTL-level signal interface.

The disk drive holds the operating system, various application programs that run on the MCC host, and copies of the terrain databases. If an 85 megabyte disk is used, there is room for three terrain databases, each covering an area of about 50 by 50 kilometers. The cartridge tape drive is used to install on the MCC system new terrain databases, which are supplied on cartridge tape. The floppy disk drive, which comes as standard equipment with the Masscomp system, is used to install updates to the operating system and application programs.

The SIMNET local area network, as described in *The SIMNET Network and Protocol*[3], is an Ethernet™ operating at ten million bits per second. Included in the MCC host is a controller card that allows it communicate via the network. The controller card, which is model ENP-30 manufactured by Communication Machinery Corporation, provides hardware support for the Ethernet physical and data link layer protocols. The card has a Motorola MC68000 microprocessor and 128K bytes of RAM that can be programmed to support protocols at higher layers.

The MCC host controls power to several individual voice radios through a TTL-level signal interface card. This card, which is model MP830-72 manufactured by Burr-Brown Corporation, provides 72 I/O lines for TTL-level

™ Ethernet is a trademark of the Xerox Corporation.

signals. Seven of these lines are used to control voice radios. The remaining lines are presently unused.

A Masscomp 5600 computer has a built-in RS-232 port for use in attaching a console terminal. The MCC host terminal, which may be any standard ASCII terminal, is connected to this port on the MCC host. An additional eight RS-232 ports are supported by an interface device, called a High Performance Serial Multiplexor (HPSM), which is supplied by Masscomp.

3.4. AppleTalk Network

An AppleTalk network links the MCC consoles with the MCC host. This network has a topology like that of an Ethernet: individual computers tap onto a backbone cable by means of transceiver devices. However AppleTalk is simpler than Ethernet, it requires less expensive hardware, and it provides less communications capacity. AppleTalk was chosen because it has sufficient capacity to support communication between the host and the consoles, and because the Macintoshes selected as consoles have built-in hardware interfaces and software support for AppleTalk.

The raw data rate on the AppleTalk network is 230.4 kilobits per second over a maximum (cable) distance of 300 meters. The cable is shielded, twisted pair. Up to 32 devices may be attached to the network. The role of each device in supporting the network is passive; should any device fail, communication among other devices on the network is unlikely to be disturbed.

There is no direct hardware connection between the MCC host and the AppleTalk network. The MCC host works through an intermediary Macintosh called the *Bridge* in order to communicate with devices across the AppleTalk network. As Figure 3-1 shows, an RS-232 port on the MCC host is linked by cable to a port on the Bridge. Another port on the Bridge, in turn, is connected to the AppleTalk network. Chapter 4 describes in more detail how this indirect connection between the MCC host and the AppleTalk network operates. In a future version of the MCC system, we plan to replace the Bridge with an interface card that plugs directly into the MCC host's Multibus backplane, connecting it to the AppleTalk network.

3.5. Software Overview

The software of the MCC system is distributed among many modules. Each of the Macintoshes in the MCC system is controlled by one of these software modules, and several modules execute simultaneously on the MCC host.

Within each Macintosh console resides the software that implements that console's user interface. Because this software is executed within the console itself, the user interface it supports can be most responsive. No communication with the MCC host slows down a console's response to its user's individual keystrokes and mouse selections.

The software within each console also implements some simpler aspects of the simulation controlled from that console. In the case of the Close Air Support Console, for example, the console software maintains the schedule of missions, and enforces the established limit on the number of sorties available. No communication with the MCC host is involved when the user simply schedules a future close air support mission. This approach permits a responsive user interface, and reduces the load on both the AppleTalk network and the MCC host.

The MCC host executes software modules that support the remaining functions of the MCC system. In general, a particular function is implemented in software on the MCC host rather than on a console Macintosh if it involves:

- communicating over the SIMNET local area network, as with combat vehicle simulators;
- floating point arithmetic calculation, such as that required to plot the locations of bomb explosions on the battlefield;
- manipulating large quantities of data, such as when examining a terrain database to determine whether a particular location is suitable for placing a CCV; or
- interaction between two or more consoles, as when the SIMNET Control Console is used to reconstitute an ammunition supply truck whose status is displayed on the Admin/Log Console.

Each element that the MCC system simulates, such as close air support or maintenance, usually involves some user interface on one of the consoles. This user interface is supported by a simple model of the simulated element, which also resides within that console. The rest of the element's simulation, which often involves more elaborate modeling and some communication over the SIMNET network, is implemented by software on the MCC host. In the case of close air support, for example, the schedule of missions is maintained within the console, but when a mission is executed the MCC host is responsible for computing the locations of bomb detonations and broadcasting those locations over the SIMNET network.

Thus each element is simulated through close cooperation between software on the MCC host and software on one or more of the consoles. Exactly what form that cooperation takes depends on which element is being simulated. In later chapters of this document, we describe, for each console, which functions are performed by software on the console, which by software on the host, and how the two communicate. Those aspects of the communication between host and console that are common to all five consoles are described in the next chapter. The remainder of this chapter describes the software environment on the Macintosh, and that on the host.

3.6. Macintosh Console Software

A program that runs on a Macintosh is called an *application*. When a Macintosh is powered on, it will load and execute an application it finds on a floppy disk inserted in its floppy disk drive. This process of loading and executing software is called *booting*.

Each Macintosh in the MCC system has a floppy disk permanently inserted in its drive, containing an application for whatever function that Macintosh is to perform. When the Macintosh is powered on, it immediately boots the application found on its floppy disk, and, as a result, becomes the SIMNET Control Console, the Admin/Log Console, or whatever. Thus each Macintosh is tailored for a specific function in the MCC system by the application contained on its floppy disk. The MCC system includes six Macintosh applications altogether: one for each of the consoles, and one for the Bridge.

The Bridge application is unlike other Macintosh applications within the MCC system in that its purpose is not to provide a user interface for some simulation. We will describe the Bridge application in the next chapter, which focuses on the AppleTalk network. The rest of this section describes, in general terms, how the five console applications work. For a detailed description of the Macintosh hardware, operating system, and Toolbox, the reader should consult *Inside Macintosh*[6].

3.6.1. Main Event Loop

The Macintosh operating system is a single–task system, meaning only one application can run on a Macintosh at any one time. Often, however, the applications that run on the Macintosh must give the illusion that they are doing many things at once. Each MCC console application must simultaneously:

- respond to the user's use of the mouse and keyboard;
- handle messages arriving from the MCC host via the AppleTalk network; and
- watch the clock for upcoming, scheduled activities, such as the arrival of a supply truck.

The Macintosh operating system queues events—such as keystrokes, mouse clicks, and the arrival of AppleTalk messages—until the application is ready to deal with them. Each application is organized around a section of code, called a main event loop, in which it selects and processes these events one by one. In a single pass through its main event loop, the application will:

- check the clock and handle any time–dependent conditions that may have arisen;

- obtain, from the Macintosh operating system, the next event to be processed; and
- process the event according to its kind.

By repeatedly cycling through its main event loop, the application is able to divide its time among all of the activities it must perform. The application's main event loop only terminates when the console upon which it runs is deactivated at the close of an exercise.

3.6.2. Event Handlers and Dialog Boxes

The most common event encountered in the main event loop is the press of a mouse button: a *mouse click event*. When the application encounters one of these events, it uses the Macintosh Toolbox to find the window within which the mouse is pointing. Because the MCC applications display a variety of windows, each window has associated with it its own *event handler* to perform the appropriate interpretation of any events, such as mouse clicks, that occur in that window. A mouse click event is passed to the event handler associated with the window clicked.

A single version of an event handler is used for most of the windows appearing in MCC applications. This event handler deals with windows, called *dialog boxes*, that contain visual devices such as buttons, check boxes, radio buttons, type-in fields, and tables. The appearance and behavior of these devices will be familiar to anyone who has used one of the MCC consoles or read *The SIMNET Master Documentation*[2]. However the names of these devices may not be as well known, so we have included Figure 3-2 showing examples of these devices labelled with their names.

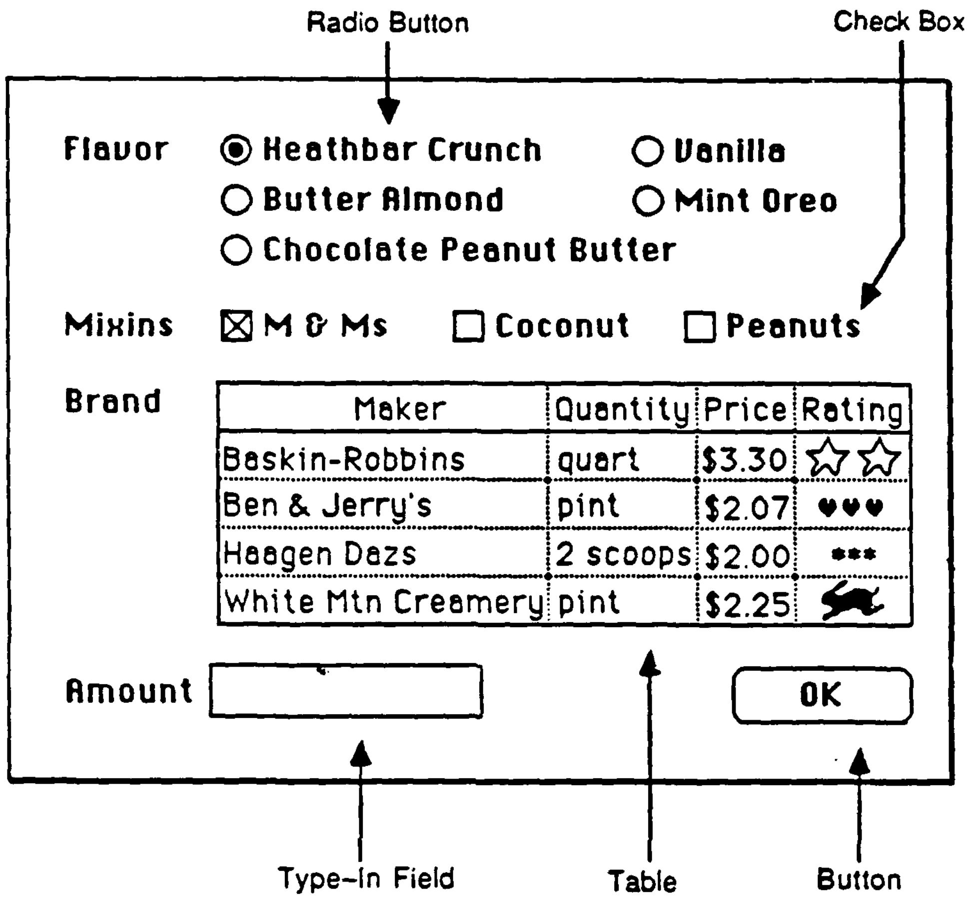


Figure 3-2. Visual devices appearing in dialog boxes.

Each of these visual devices has a consistent behavior wherever it is used in a window. For example, check boxes always appear either "on" or "off", and toggle between those states when clicked. The dialog box event handler implements this behavior for check boxes (with help from the Macintosh Toolbox), and thereby ensures that check boxes behave the same wherever they are found.

Although the visual devices behave consistently from one dialog box to another, no two dialog boxes contain exactly the same arrangement of these devices. Each dialog box is uniquely constructed with parts from the single, common set of visual devices. The dialog box event handler knows how to implement these visual devices in general, but it also must have information on how those devices are applied in each specific case. Specific information about each dialog box is contained in a data structure called a *dialog box definition*.

The definition for a dialog box is a static object created by the programmer. Subsequently, when the application is running, the definition is used to dynamically create the dialog box window on the screen at the appropriate moment. While that window is visible the window and its definition remain coupled.

The definition is used to draw the dialog box on the screen, to supply initial values for the fields of the dialog box, and to record any information that is entered using the dialog box. It is also used to handle the events that pertain to the dialog box. When a mouse click event occurs, for example, knowledge of the window clicked upon leads the event handler to the appropriate definition needed for processing the event.

The dialog box definition specifies what visual devices appear in the dialog box, and where. It also contains information permitting the dialog box event handler to:

- validate whatever the user enters into a particular type–in field. The field can be checked to ensure that it contains a number within a certain range, that it contains a UTM grid coordinate of a certain precision, that it contains a date time group, or that it contains no more than a certain number of characters.

- ensure that the user has filled in a particular type–in field before leaving the dialog box.

- group several radio buttons so that only one button of the group is on at any one time.

- invoke a programmer–supplied routine to perform any special drawing in the dialog box or custom processing of events pertaining to the dialog box.

Another common event encountered in the main event loop is the press of a key on the keyboard: a *key down event*. By user interface convention, keystrokes always pertain to the frontmost window on the screen. Therefore when the application encounters one of these events, it passes it to the event handler associated with the frontmost window.

When the user clicks on a type–in field with the mouse, that field becomes the *current field* of the dialog box. The current field is always indicated by being highlighted, or by the presence of a blinking cursor within that field. The dialog box event handler keeps track of the current field. It applies all key down events to the current field, so that any characters typed by the user are inserted at the blinking cursor.

The user may attempt to make a new field the current field, either by clicking elsewhere in the dialog box or by pressing the Tab key on the keyboard. Before letting the user move to a new field, however, the dialog box event handler checks whatever was typed in the old field for validity. The dialog box definition may specify constraints on the contents of a particular type–in field, such as that it contain a number within a certain range. The event handler applies these constraints and displays an error message if the constraints are not met.

3.6.3. Network Events

Another kind of event encountered in the main event loop is a *network event*, announcing the arrival of a message from the AppleTalk network. Each message sent through the MCC system's AppleTalk network bears a code indicating its meaning. This code is used by the application to determine how to respond to the message, and to interpret any other data contained in the message. In the following chapter we will describe the AppleTalk network in more detail, and in subsequent chapters we will mention each of the different types of messages communicated over the AppleTalk network.

3.6.4. Development Environment

The applications for the MCC consoles are written in the C language. Early development was done using a cross–development environment, hosted on a UNIX workstation, called the Stanford UNIX Macintosh C Development Kit (SUMacC). This development environment was created and shared by the Stanford University Medical Center. More recent development has used a standalone C development environment for the Macintosh produced by THINK Technologies, Inc.

3.7. Host Software

The software on the MCC host is divided into several modules. Each module exists as a separate process on the host, there to perform a specific function.

3.7.1. Processes

The processes are generally of two sorts. Those referred to as *console processes* serve as host counterparts to the MCC consoles. One console process is dedicated to supporting each console. The remaining processes are called *service processes*. They perform a variety of different functions in support of the console processes.

Figure 3–3 is a schematic view of the processes and some other software objects within the MCC host. The processes are shown with names like SCC, Admin, and ATRecv, and they are grouped according to whether they are console processes or service processes. The other objects, with names like MCC Parameters and Semaphores, allow communication between the processes. These communication objects will be described shortly.

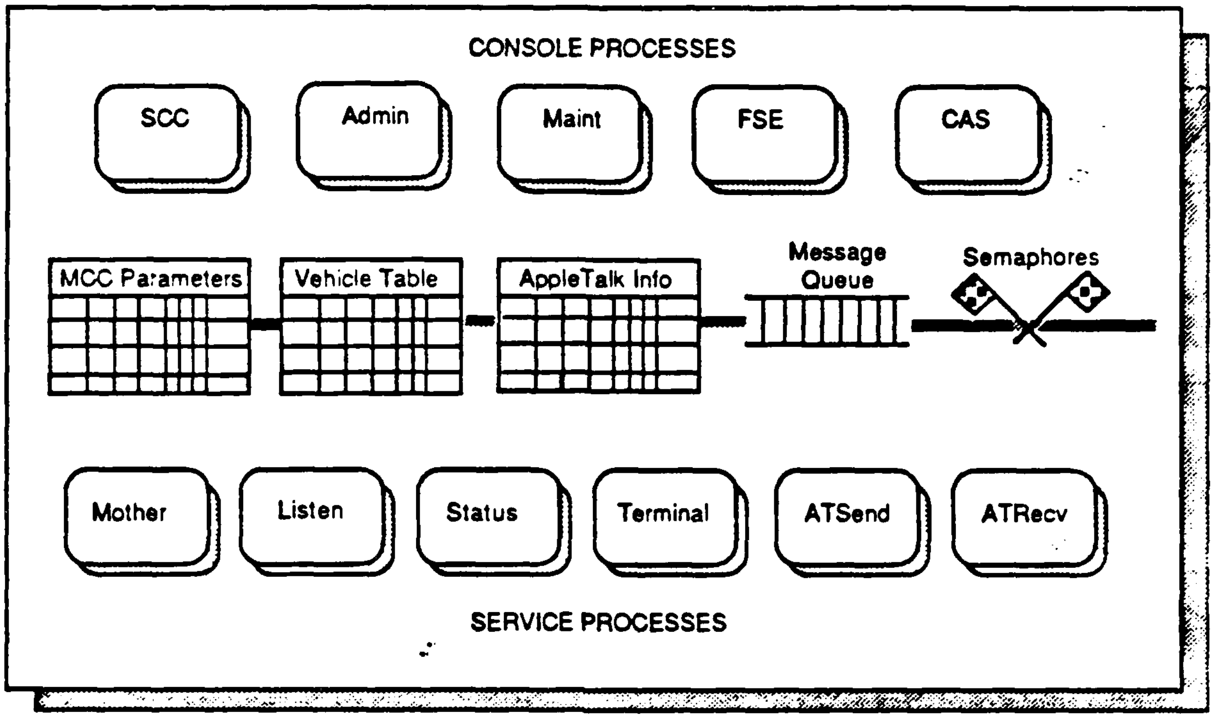


Figure 3–3. Host software environment.

These are the console processes:

SCC is the host counterpart to the SIMNET Control Console application.

Admin is the host counterpart to the Admin/Log Console application.

Maint is the host counterpart to the Maintenance Console application.

FSE is the host counterpart to the Fire Support Console application.

CAS is the host counterpart to the Close Air Support Console application.

Each console application on its Macintosh is in one–to–one communication with the corresponding console process on the host.

These are the service processes and the purposes of each:

Mother performs a wide variety of functions, including initializing the MCC host software at the start of an exercise, positioning vehicles on the battlefield, modeling CCVs, computing the positions of bomb detonations, activating combat vehicle simulators, and recording the statistics collected from an exercise.

Listen listens to the SIMNET local area network and processes the information received from it.

Status monitors the health of the MCC system and periodically reports its findings, via the SIMNET local area network, to the local Network Operations and Maintenance system.

Terminal supports the host terminal, and processes commands entered at that terminal.

ATSend formats messages going from the host to the AppleTalk network.

ATRecv unformats messages coming to the host from the AppleTalk network.

The Mother process is the first to run when the MCC system is started. It proceeds to start each of the other service processes, then the SCC process. In the course of the exercise various optional functions, such as the *fire support* element and close air support, may be initialized from the SIMNET Control Console. As each optional element is initialized, the SCC process starts the particular console process needed to support that element. When the fire support element is initialized, for example, the FSE process is started. Figure 3-4 summarizes which processes start which other processes.

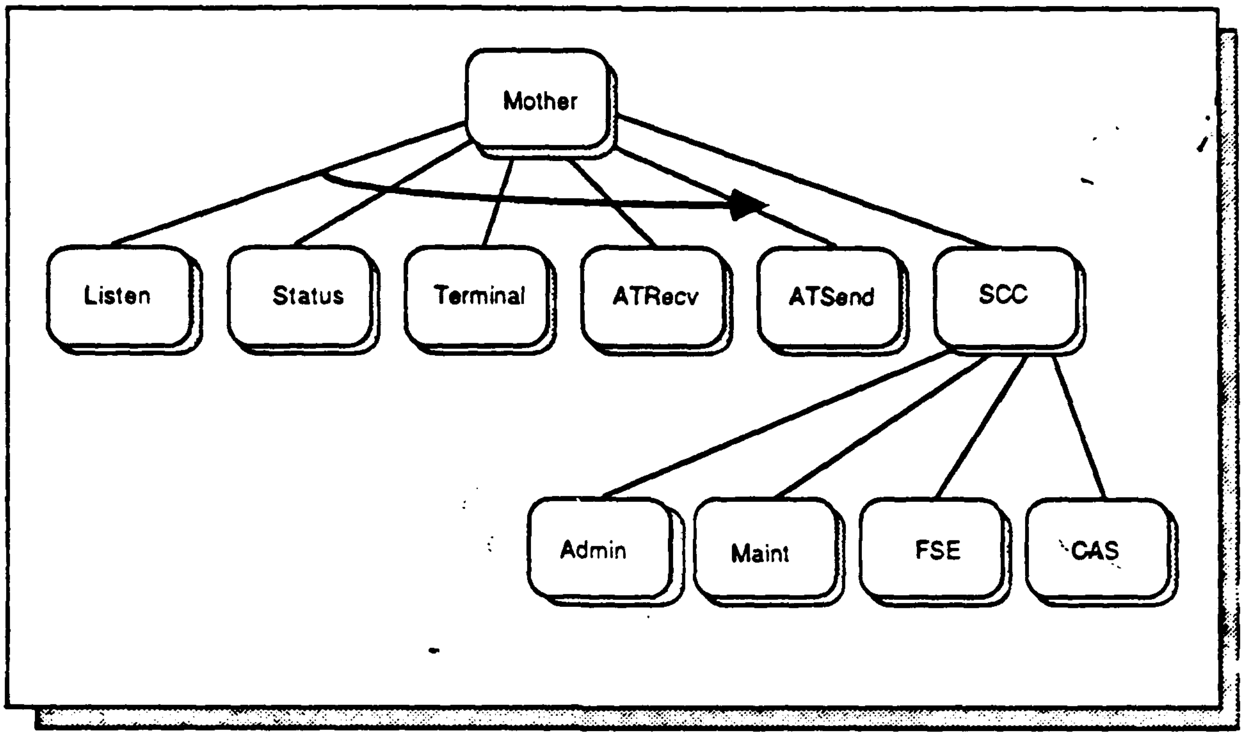


Figure 3–4. Process invocation hierarchy.

Dividing the host software into these processes affords two advantages relevant to the architectural goals of the MCC system. First, because each console process is somewhat isolated from its peers, development on that console can proceed independently of development on other consoles. This facilitates evolutionary development—"rapid prototyping"—of the MCC system. Second, the system can take advantage of a multiprocessor host computer because two processes can be executed simultaneously by an MCC host computer with two processors. The Listen process, for example, might be dedicated to one processor, while the remaining processes share the other processor. Dividing the processes among two processors in this way, if necessary, would allow the MCC system to handle a greater workload.

The remainder of this section addresses topics applicable to all of the host processes. In subsequent chapters, we will consider the processes individually, and discuss certain things specific to each.

3.7.2. Interprocess Communication

While the MCC system is operating, most or all of the host processes are active. The processes must cooperate closely in a number of ways to coordinate the overall behavior of the MCC system. That means, of course, that they must communicate.

The UNIX operating system for the Masscomp 5600 computer provides several mechanisms by which processes may communicate. The MCC system uses three of these mechanisms: *shared memory*, *semaphores*, and a *message queue*.

Shared memory is a data structure in main memory, to which two or more processes have direct access for reading and modifying. Care must be taken to ensure that, when one process is modifying the common data structure, the other processes do not try to read it, lest they find the data structure in an inconsistent state.

Semaphores may be used to coordinate access to a common data structure so that only one process uses the data structure at one time. A semaphore is like a flag that a process can wave while it is using the data structure. The operating system guarantees that only one process can have the flag at one time. If one process already has the flag because it is using the data structure, but another process tries to get the flag, the second process will be made to wait until the first one is done.

A message queue is used by processes to send messages to each other. Any process can send a message to any other at any time. Each message can bear some code indicating its meaning, and some quantity of data or parameters. A process which is expecting to receive messages can either periodically check to see if any are queued for it, or stop and wait until a message arrives.

The MCC host uses three blocks of shared memory for communication among processes. They are called *MCC Parameters*, *Vehicle Table*, and *AppleTalk Info*. The MCC host also uses a message queue, through which any process may send a message to any other, and a set of semaphores for coordinating access to shared memory. In Figure 3-3 these objects are shown providing communication between processes of the console layer and those of the service layer. The same objects provide communication among the processes within each of these layers, as well.

3.7.3. MCC Parameters Shared Memory

The MCC Parameters shared memory contains a variety of information of interest to various processes. There is no simple rule governing which processes generate or use this information; it comes from many sources and is used in many places. MCC Parameters includes:

- the number of combat vehicle simulators on the SIMNET local area network, available for the exercise;
- the exercise identifier to be used on the SIMNET local area network, for the exercise;
- the address on the SIMNET local area network of the local Network Operations and Maintenance system;

- the name of each console application, as it is known on the AppleTalk network. (In the following chapter we will describe how these names are used.)
- the number of terrain databases and the location of each on the MCC host's disk drive;
- whether the the exercise has started, and, if so, when;
- which optional elements (such as the fire support element and the stealth jeep) have been indicated as being allowed to participate in the exercise;
- which companies are playing in the exercise, and on which side each company is playing;
- which battlefield terrain has been chosen for the exercise, and information about that terrain including the dimensions of the battlefield and the mapping between UTM grid coordinates and locations on the battlefield
- the locations of the ammunition and fuel depots in the brigade support area, and the location of the unit maintenance collection point;
- information about each supply truck and maintenance team's truck, including where the truck is located, what load is on board (if it is a supply truck), and to which company and force the truck is assigned;
- the locations and azimuths of fire of the howitzer battery and mortar platoon, and the quantities of ammunition on hand at each; and
- the number of close air support sorties allotted for the current day, and the number of those that may be preplanned.

3.7.4. Vehicle Table Shared Memory

The Vehicle Table shared memory contains information about the combat vehicles the MCC system initializes and the computer-controlled vehicles it simulates. For each combat vehicle, this table records the following information about the combat vehicle and its simulator:

- the physical location of the simulator;
- the simulator's address on the SIMNET local area network;
- the type of combat vehicle (e.g., M1 Abrams main battle tank) simulated by the simulator;

- whether the combat vehicle has been placed on the battlefield by the MCC system;
- whether the simulator reports that it is actively simulating its combat vehicle;
- the company to which the combat vehicle has been assigned, and the bumper number identifying it within that company;
- whether the combat vehicle is assigned to the offense force or the defense force;
- the combat vehicle's location, orientation, and appearance as last reported by its simulator via the SIMNET local area network;
- the combat vehicle's maintenance and supplies status as last reported by its simulator via the SIMNET local area network; and
- cumulative totals of the distance travelled and supplies used by the combat vehicle.

For each computer-controlled vehicle simulated by the MCC system, the Vehicle Table records:

- what type of vehicle the CCV is;
- which host process is responsible for the CCV's existence. Each CCV is the creation of some process. The FSE process, for example, creates the CCVs representing the self-propelled howitzers and the mortar carriers.
- whether the CCV is visible on the battlefield, and, if so, whether it is healthy, on fire, or a blackened wreck;
- if the CCV is on fire, how long it has been burning. (CCVs are made to burn for 15 minutes.)
- if the CCV is visible, its location, orientation, and appearance on the battlefield; and
- if appropriate, which company the CCV has been assigned to.

The status of either a combat vehicle or a computer-controlled vehicle may be quickly located in the Vehicle Table by its vehicle number.

3.7.5. AppleTalk Info Shared Memory

The AppleTalk Info shared memory supports the host's AppleTalk interface. It is used for communication between the console processes, which send and receive AppleTalk messages, and the ATSend and ATRecv processes, which convey those messages to and from the Bridge. The following chapter describes this in detail.

3.7.6. Message Queue

The message queue is used by all of the processes in order to signal events to, and request services of, other processes. About twenty different kinds of messages are used. Each message bears the identities of its source and destination processes, a code indicating what kind of message it is, and, depending on the kind of message, some data or parameters. Each message is mentioned at the appropriate points in the remainder of this document where the consoles and service processes are discussed. Moreover all of the various kinds of messages are summarized in Appendix B.

There are two sorts of interactions a pair of processes may engage in via the message queue: one-way interactions and round-trip interactions. In a one-way interaction, a process sends a message to another and does not expect a response. This form of interaction is usually used when one process must report an event to another. In a round-trip interaction, a process sends a message to another, then waits for a responding message. This form of interaction is usually used when one process must request a service from another, and needs to know the results of its request. Wherever we discuss a particular kind of message, we will indicate whether it is part of a one-way or round-trip interaction.

3.7.7. Process Behavior

Each of the MCC host processes spends much of its time in a loop that is comparable to a Macintosh application's main event loop. During one cycle through its loop, a process will wait for an event signalling the arrival of new work, examine the event that occurs, and handle it appropriately. The work may come from any of three sources:

- as a message sent from another process via the message queue;
- as a protocol data unit (PDU) received from the SIMNET local area network; or
- as data received through one of the RS-232 ports.

Each process looks for events from just one of these three sources. The Listen process awaits PDUs from the network. The Terminal and ATRecv processes await input from the RS-232 ports leading to the host terminal and Bridge, respectively. All other processes await messages from other processes.

Besides handling events from one of these sources, most processes are also responsible for some time-related activities. For example the Mother process, which models CCVs, must periodically broadcast VehicleAppearance PDUs describing those CCVs.

A process can schedule a time-related activity by setting an *alarm* to expire some seconds in the future. With the alarm set, the process can do other processing without having to watch a clock. When the alarm expires, the process is interrupted so that it can perform the scheduled activity. For simplicity, alarms are only allowed to interrupt a process while that process is awaiting an event such as the arrival of a message. Alarm interrupts are suppressed while the process is handling an event or another alarm interrupt.

The Mother process provides one example of the use of alarms. It sets an alarm every second to prompt itself to send some VehicleAppearance PDUs describing the CCVs it is modeling. When not responding to these alarms, the Mother process is awaiting, accepting, and processing messages sent to it via the message queue.

3.7.8. Ethernet Interface

The MCC host has an interface to the Ethernet that serves as the SIMNET local area network. The protocol data units (PDUs) described in *The SIMNET Network and Protocol*[3] are sent and received on this network.

Software controlling the Ethernet interface resides in three places:

- within the host's operating system kernel, in the form of a device driver;
- within each process that sends or receives Ethernet packets, as library routines incorporated into that process; and
- within the Ethernet controller card, as a program loaded into the card and run by its resident microprocessor.

Packets received by the Ethernet controller card are buffered locally, then copied to buffers located in the host's main memory. A single process running on the host, the Listen process, obtains packets from those buffers and processes the PDUs found within them.

PDUs are sent or broadcast on the Ethernet by several different MCC host processes. These processes place their PDUs, as packets, within buffers located in the host's main memory. From there, the Ethernet controller card copies the packets to its own, local buffers, then transmits the packets onto the Ethernet.

3.7.9. TTL-Level Signal Interface

Using a TTL-level signal interface, the MCC host controls power to several voice radios that are used for command and control of the battalion. Using the memory mapping facilities of the host operating system, the control registers on the interface card are assigned addresses as though they existed in the host's main memory. These registers are then addressed directly by the SCC process to influence the TTL-level output lines, thus enabling and disabling individual radios.

3.7.10. Development Environment

All MCC host software is written entirely in the C language. Development was done on Masscomp 5500 and 5600 computers, under Masscomp's Real-time UNIX (RTU™) operating system.

3.8. MCC Configuration File

An MCC system is tailored to the needs of a particular SIMNET installation by a file that resides on the MCC host. This file is called the MCC configuration file. It is read when the MCC system is started to determine, among other things, what combat vehicle simulators are available at the installation, and what terrain databases are stored on the MCC host's disk drive. Appendix A of this document fully specifies the content and format of this file.

™ RTU is a trademark of the Massachusetts Computer Corp.

4. APPLETALK NETWORK

This chapter describes the AppleTalk network, and how it is implemented and used by the MCC system. AppleTalk is fully defined in an Apple publication entitled *Inside AppleTalk*[7], but we have included here a brief overview of those parts of AppleTalk relevant to the MCC system. This is followed by a description of the Macintosh and MCC host implementations of AppleTalk. Finally, we discuss the interaction between a console and its counterpart console process on the MCC host, and how the AppleTalk network supports that interaction.

4.1. AppleTalk Protocols

The AppleTalk network developed by Apple Computer involves not only a physical wire for carrying data, but also a suite of protocols for use on that wire. The protocols are organized into a hierarchy of layers, like that prescribed by the International Standards Organization's (ISO) Open System Interconnection (OSI) basic reference model. Each layer of protocol provides services that are used, in turn, by the next higher layer to provide services that are yet more elaborate or specialized. The lowest layers are concerned with grouping data into packets, addressing packets to specific nodes on the network, and detecting transmission errors. Intermediate layers build on the lowest layers to provide reliable data transfer, and to provide a means for locating entities on the network by name. The highest layers contain protocols for specific applications, such as controlling printers or accessing file servers.

The computers attached to an AppleTalk network are called *nodes*. Within each node there are one or more logical entities, known as *sockets*. Nodes, and the sockets within a node, are addressable from other nodes on the network.

The MCC system directly uses the services of two AppleTalk protocols: the *Name Binding Protocol (NBP)* and the *AppleTalk Transaction Protocol (ATP)*. These protocols, in turn, use lower layer protocols called the *Datagram Delivery Protocol (DDP)*, and the *AppleTalk Link Access Protocol (ALAP)*. The relationship between these protocols is depicted in Figure 4-1.

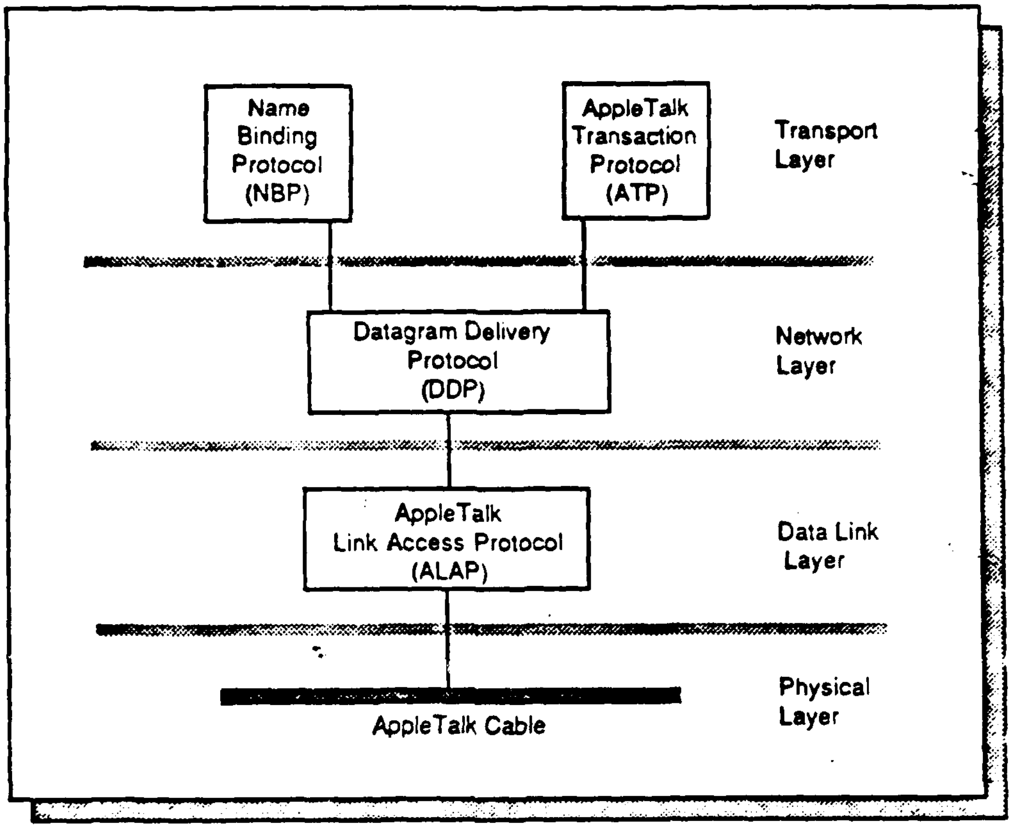


Figure 4-1. AppleTalk protocols used by the MCC system.

NBP allows the sockets within a node to be known and located by symbolic names rather than numeric addresses. This is an important service because nodes do not have fixed addresses on an AppleTalk network. Each time a node is powered on, it invents an address, and, after checking that no other node on the network already has that address, adopts the address for itself. The SIMNET Control Console, for example, may have a different address every time it is powered on, so the MCC host cannot always locate it at a particular, known address on the network. With NBP, however, the MCC host can ask if there is anything on the network with the name "SCC". The SIMNET Control Console, recognizing its name, will then respond, providing its current address.

ATP provides a reliable, transaction service between sockets. Using ATP, a client attached to one socket may send a request to a server at another, and receive a response back. The request and the response may each contain several hundred bytes of data. ATP attempts to provide a reliable service by resending the request or response to overcome any transmission errors or losses. Both the client and the server may have confidence that ATP will do the best job possible to ensure that a transaction is completed successfully.

Both NBP and ATP are supported by the services of DDP. DDP conveys individual packets of data, called *datagrams*, between any two sockets on the network. DDP can also be used to broadcast a datagram to a particular number of socket within every node.

DDP, in turn, makes use of ALAP, which moves packets of data, called *frames*, between nodes on the network. Because the network is a single cable shared by all nodes, ALAP coordinates use of the network so that nodes don't interfere with each other when transmitting.

ALAP operates directly on the physical AppleTalk cable, which is a shielded, twisted pair. Signalling on the cable is at 230.4 kilobaud, using standard RS-422 drivers and receivers.

4.2. Macintosh Implementation

Apple Computer supplies Macintosh software supporting several AppleTalk protocols, including the four protocols needed by the MCC system. Physical communication on the AppleTalk cable is implemented by hardware that is a standard part of every Macintosh.

A Macintosh application that is expecting to receive ATP requests can supply, in advance, buffers that will be used to contain those requests when they arrive. The application is notified when a request arrives by an event it encounters in its main event loop. Similarly, an application that is expecting to receive a response to a request it is issuing can supply, in advance, a buffer to contain that response. An event will signal the arrival of the response. All of the consoles make use of this feature, which allows them to engage the MCC host in ATP transactions while carrying on with other processing.

4.3. Host Implementation

The MCC host is able to participate on the AppleTalk network because of several components: a Macintosh computer programmed to become the Bridge; an RS-232 line between the host and the Bridge; host processes called ATSend and ATRecv; the AppleTalk Info shared memory on the host; and a library of routines incorporated into each host console process. Figure 4-2 shows how these components are related, and where each protocol is implemented among them.

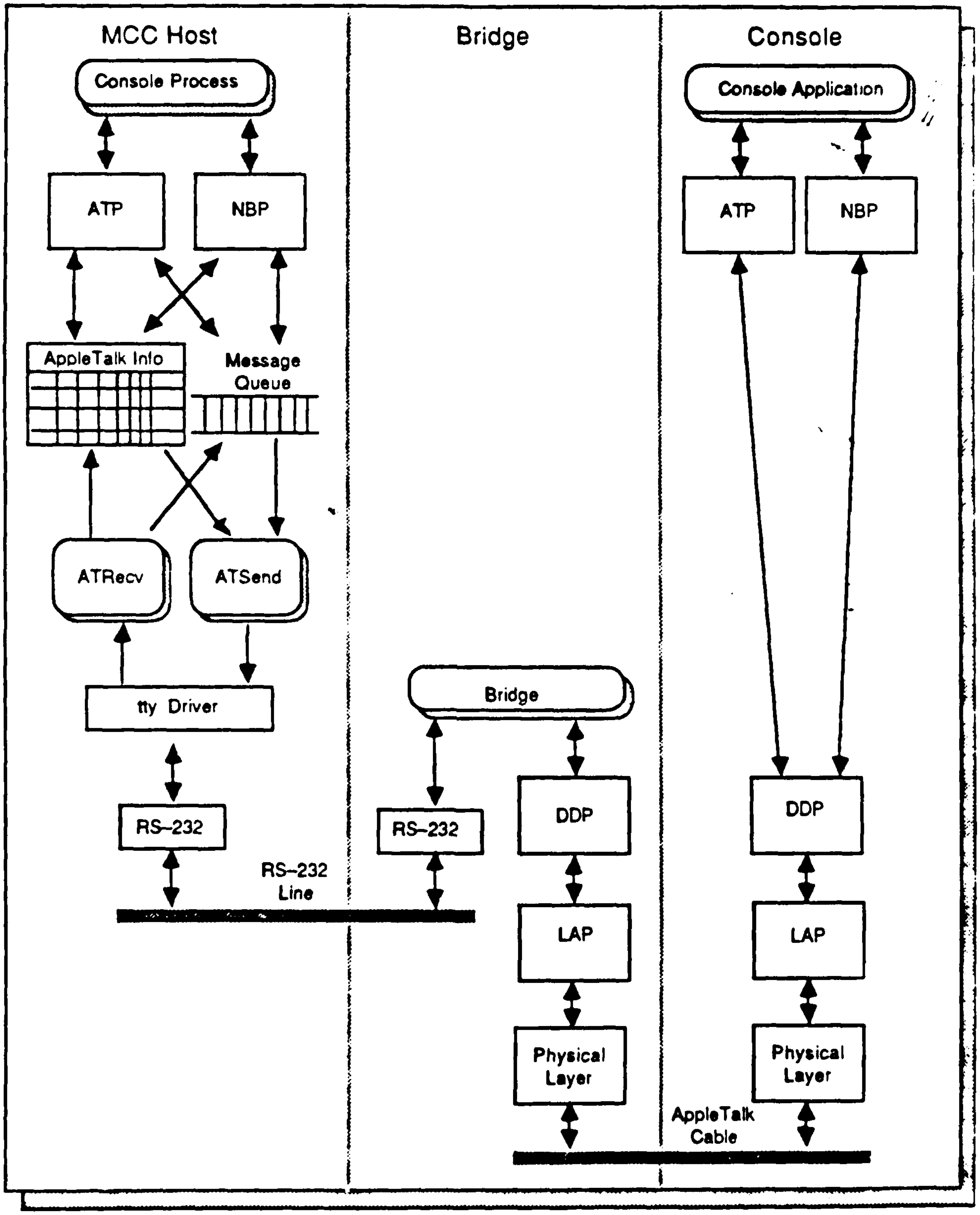


Figure 4-2. AppleTalk implementation.

Whereas a packet communicated over the AppleTalk network may contain any data, including binary integers and character strings, information transferred over the host's and Macintosh's RS–232 ports is most conveniently expressed as a sequence of ASCII characters terminated by a "newline" character. Therefore AppleTalk packets transferred to and from the host are encoded as ASCII characters so that they can be passed over the RS–232 line between the host and the Bridge. The encoding scheme we have chosen is one called *byte–stuffing*, in which:

- any newline character appearing in the packet is replaced by an "escape" character followed by a distinguished character, such as a zero;
- any escape character appearing in the packet is replaced by two escape characters; and
- the packet is terminated by a newline character.

A packet encoded in this way is decoded by:

- replacing all instances of an escape character followed by a zero with a single newline character;
- replacing all instances of two successive escape characters with a single escape character; and
- removing the terminating newline character.

The RS–232 line between the host and the Bridge uses this byte–stuffing encoding scheme, and operates at 9600 baud.

4.3.1. Bridge

The Bridge acts as an intermediary between the MCC host and the AppleTalk network. It takes advantage of DDP and ALAP software supplied with the Bridge Macintosh, and the Macintosh hardware that implements the AppleTalk physical signalling, so that the host can participate on the AppleTalk network.

Since the Bridge is the MCC host's representative on the AppleTalk network, the host's address as a node on the AppleTalk network is simply that of the Bridge. When the MCC system begins executing, the host discovers its AppleTalk address by querying the Bridge for it.

ATSend and ATRecv are the host processes that communicate directly with the Bridge. They transfer DDP packets to and from the Bridge, via the RS–232 line.

4.3.2. Sending Datagrams

Any console process on the host may send a DDP packet. To do so, it obtains one buffer from a pool of buffers contained in the AppleTalk Info shared memory, and places the data portion of the packet in this buffer. It then sends an *ATalkSend* message via the message queue to the ATSend process. This message, which represents a one-way interaction between the processes, identifies the buffer containing the packet data, and the address of the packet's recipient. When ATSend receives this ATalkSend message, it encodes the packet using the byte-stuffing scheme, includes source and destination addresses in the packet's header, and sends the encoded characters to the Bridge. The Bridge receives the packet, decodes it, and transmits it on the AppleTalk network as a DDP packet.

4.3.3. Receiving Datagrams

The Bridge receives those DDP packets from the AppleTalk network that bear a broadcast address or the address of the Bridge node. When a packet is received, the Bridge encodes it using the byte-stuffing scheme, includes source and destination addresses in the packet's header, and sends the encoded characters to the host. ATRecv receives the encoded characters, decodes them, and places the data portion of the DDP packet in a buffer selected from the AppleTalk Info buffer pool.

The AppleTalk sockets on the host are represented by a table contained in the AppleTalk Info shared memory. Any host process wishing to be associated with a particular socket, and receive any packets addressed to that socket, may identify itself with the entry in the socket table corresponding to that socket.

A DDP packet received by ATRecv bears, as its destination, the address of a particular socket. ATRecv looks in the socket table to determine which console process is associated with that socket. If none if found, the packet is ignored. Otherwise, ATRecv sends an *ATalkRecv* message to the console process, notifying it of the packet's arrival. This message, which represents a one-way interaction between the processes, bears the source and destination addresses of the packet, and it identifies the buffer containing the data portion of the packet.

4.3.4. ATP and NBP

ATP and NBP are implemented on the host as a collection of library routines, a copy of which is incorporated into each console process. These routines are called to send and receive ATP requests and responses, and to locate network entities by name using NBP. The routines implement these protocols by sending and receiving DDP packets using ATSend and ATRecv.

4.4. Console Operation

The SIMNET Control Console becomes available as soon as the MCC system is started. The other consoles are made available as the elements they control are initialized. In each case, a console is activated by starting its corresponding console process on the host, and that process then activates the console Macintosh.

One of the first things a console process on the host does is to establish communication with its Macintosh application. Since the process cannot know the AppleTalk node address of the Macintosh (which varies, as explained earlier in this chapter), the process locates the Macintosh using NBP. Each console application has a particular name, unique among all entities on the AppleTalk network. In the case of the SIMNET Control Console, for example, this name is "SCC". From the time a console Macintosh is powered on, its application listens to the AppleTalk network for that name. When the console process on the host is ready to contact its application, it uses NBP to look for the application by name. The application responds and supplies its current address.

Using this address, the console process then sends an ATP request telling the application to make itself available as a console. In this request, the process supplies the current date and time so that the Macintosh's clock can be set correctly. In most cases, this request is followed by additional ATP requests containing information about the mapping between UTM grid coordinates and locations on the battlefield, for the terrain database chosen. This mapping information allows the application to accept and process coordinates entered by its user. Finally, the process will send additional ATP requests containing initialization data specific to the console. The Admin/Log application, for example, is sent the location and load of each supply truck, and the locations of the depots.

From the time it is powered on until the time it receives the last of these request, the console application displays a dialog box indicating that it is not yet available for use. When it has received the last in the series of initialization requests, this dialog box is replaced with the console's normal display.

Communication between the console process and the console application, from that point on, is in the form of ATP transactions. Some transactions are initiated by the host process, others by the Macintosh application, so both parties remain listening for incoming requests. Most of these transactions are quite specific in nature, and pertain only to a specific type of console. For example, between the Macintosh application and host process of the Close Air Support Console, there is a transaction meaning "drop some bombs".

A few types of transactions are supported by all consoles. These are:

- a request by the host process to set the Macintosh's clock to a particular date and time. This is used to implement the host terminal command for changing the MCC system's clock during an exercise; and

• a request by the host process for the Macintosh to reset itself, as though powered off and on.

At the close of an exercise, each console process sends a request to its corresponding Macintosh application to reset the Macintosh. This causes each Macintosh to boot, as though just powered on, and become ready for a new exercise.

5. SERVICE PROCESSES

This chapter describes two of the service processes that reside in the MCC host: the Mother process, and the Listen process. They are called service processes because they provide a foundation of services that support the console processes. These services include positioning vehicles and detonations on the battlefield, modeling computer-controlled vehicles, and monitoring the SIMNET local area network.

5.1. Mother Process

The Mother process is the ancestor of all other processes on the MCC host. As such, it establishes the software environment on the MCC host and starts the other processes. At the end of a simulation exercise, the Mother process reverses these steps to prepare for a new exercise. The Mother process is also the provider of the greatest variety of services to the console processes. This services it provides are positioning vehicles and detonations on the battlefield, modeling CCVs, activating combat vehicle simulators, and collecting and recording statistics from the exercise.

5.1.1. Initialization

The Mother process is the process executed from the UNIX environment to start the MCC system. It is therefore responsible for the MCC system's initialization. This initialization includes creating the interprocess communication objects used by the MCC system, processing the MCC configuration file, and starting a number of other processes executing on the MCC host.

Most of the information from the MCC configuration file is copied into shared memory by the Mother process so that other processes on the MCC host will have access to it. Information about combat vehicle simulators is placed in the Vehicle Table shared memory; other information, including the list of terrain databases on the MCC host's disk drive, is placed in the MCC Parameters shared memory.

Some time later, the exercise will actually be started by the BattleMaster or battalion S3 from the SIMNET Control Console. When this happens, the SCC process notifies the Mother process of the event. The Mother process then opens the terrain database selected for the exercise, and initializes a memory-resident cache of terrain information. This cache is described in the following section. A data structure called the *CCV Population Density Map* is also initialized. This data structure is described in section 5.1.3, which discusses the Mother process's modeling of CCVs. Finally, an entry is recorded in a file on the MCC host, logging the date and time at which the exercise was started.

5.1.2. Terrain Database

The MCC host has, on its disk, a copy of each terrain database available for exercises. Each database is represented by four files:

- a file of terrain data organized as polygons representing the surface of the battlefield, and prisms having quadrilateral cross-section and nonzero height, representing fixed objects on the battlefield;
- a file of indices into the polygon file, where each index points to the data for a 250 by 250 meter patch of terrain;
- a file describing the mapping between UTM grid coordinates and locations on the battlefield; and
- a file of miscellaneous information, such as the dimensions of the area covered by the terrain database, and the names of map sheets corresponding to that area.

The Mother process is the only process requiring access to the polygon and index files because all requests to position objects on the terrain—be they vehicles or shell explosions—are handled by that process. The entire index file, which is not large, is loaded into main memory for quick access. The Mother process also caches up to fifty terrain patches (of 250 by 250 meters) in main memory to reduce the number of time-consuming disk accesses needed. Patches are read from the terrain database as needed, and cached in main memory. If a terrain patch is needed that is not already in the cache, and the cache is full, the least recently used patch in the cache is removed to make room for the new one.

When a patch is placed in the cache, some preliminary processing is done on the patch, and the results of this processing are cached also. First, the 250 by 250 meter patch is subdivided into four patches of 125 meters square, so that, on the average, only one fourth as many polygons will have to be considered when locating the polygon at a particular point on the terrain. Second, a list of edges is built for each polygon and prism, so that subsequent tests of whether a particular point is within any of the polygons may proceed quickly. The format of a terrain patch, and the algorithms used to process a patch, are the same for the MCC system as those described in *SIMNET M1 Abrams Main Battle Tank Simulation*[8].

5.1.3. Computer Controlled Vehicles

The Mother process performs a number of tasks to model computer-controlled vehicles. When a CCV is visible on the battlefield, the Mother process periodically broadcasts a VehicleAppearance PDU describing the appearance of that CCV. This appearance changes when the vehicle is destroyed, or, in the case of a mortar carrier or self-propelled howitzer, when a muzzle flash is produced by the firing of a shell. When a CCV becomes invisible, the Mother process broadcasts several Deactivate PDUs, and ceases broadcasting VehicleAppearance PDUs, for that CCV.

The Mother process models each CCV as a system with five states, as shown in Figure 5–1. At any point in time, any CCV is in one of these five states. The Mother process's treatment of a CCV depends on its state. A CCV changes state as a result of some event, like being destroyed. These are the five states, and their meanings:

healthy	the CCV is visible on the battlefield
burning	the CCV is visible, but it has been destroyed and is on fire
destroyed	the CCV is visible, and has been destroyed, but is no longer on fire
disappearing	the CCV has just become invisible
invisible	the CCV is not visible on the battlefield

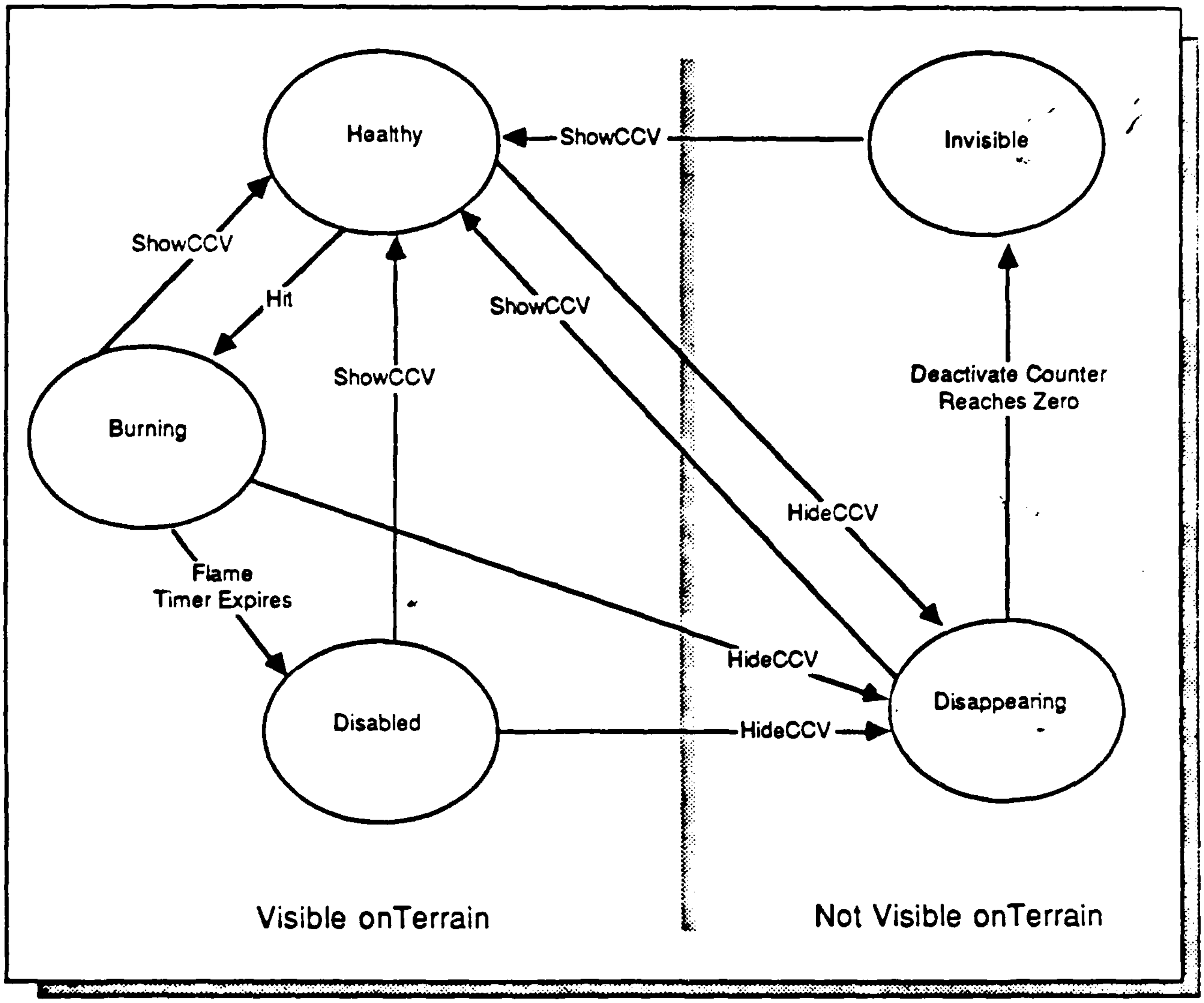


Figure 5–1. States of a CCV.

Each second, the Mother process selects a subset of all the CCVs. The subset is chosen on a rotating basis, so that each CCV will be selected at least once every five seconds. Moreover any CCV is included in the subset if its state has changed during the previous second.

For each CCV in the subset, the Mother process may perform a particular action that depends on the state of that CCV:

healthy	a VehicleAppearance PDU is broadcast, showing the vehicle healthy
burning	a VehicleAppearance PDU is broadcast, showing the vehicle burning and disabled

destroyed	a VehicleAppearance PDU is broadcast, showing the vehicle disabled
disappearing	a Deactivate PDU is broadcast for the vehicle
invisible	no action is taken

A number of things can happen during the course of an exercise to change the state of a CCV. At the beginning of an exercise, all CCVs are in the invisible state. They become visible as they are placed on the battlefield in their initial positions. They become invisible again when moved, and they begin burning when damaged by direct fire, indirect fire, or a collision with another vehicle.

These are the events that change a CCV's state:

ShowCCV ShowCCV is a message which is sent to the Mother process by another process that wants a CCV to become visible. It is used when a CCV is first placed on the battlefield, when it arrives at a destination, and when it is reconstituted. When this message is received, the CCV enters the healthy state regardless of its previous state (unless the message requests that the CCV be shown at a location outside the bounds of the terrain database, in which case the ShowCCV message is treated as a HideCCV message).

HideCCV HideCCV is a message sent to the Mother process by another process that wants a CCV to become invisible. When this message is received, the CCV enters the disappearing state (unless the CCV was already in the invisible or disappearing states, in which case the message is ignored).

hit When a CCV in the healthy state is hit by direct or indirect fire, or when a combat vehicle collides with it, that CCV enters the burning state.

timer expires When a CCV enters the burning state, a 15 minute timer is started. When this timer expires, the CCV enters the disabled state.

counter zero When a CCV enters the disappearing state, a counter is set to the number of Deactivate PDUs that must be transmitted. When the counter reaches zero, the CCV enters the invisible state.

There are two common situations in which the Mother process needs to know the locations of CCVs in a given area. First, when the Mother process places a vehicle on the battlefield, it checks to make sure that the location at which the vehicle is to be placed is not already occupied. Second, when the Mother process computes the locations of shell and bomb explosions, it determines whether or not any CCVs are destroyed by the explosions. While the location of each CCV is available in the Vehicle Table shared memory, it would be inefficient to check the location of every CCV in each of these situations.

To make the search for CCVs in a given area more efficient, the Mother process creates a data structure that divides the terrain into 500 by 500 meter squares called *buckets*. Each bucket contains a list of all the CCVs in that square of terrain. This structure, called the *CCV Population Density Map*, is represented pictorially in Figure 5-2. When a CCV becomes visible on the battlefield, that CCV is added to the list of CCVs in the appropriate bucket. The CCV is removed from the bucket when it disappears from the battlefield.

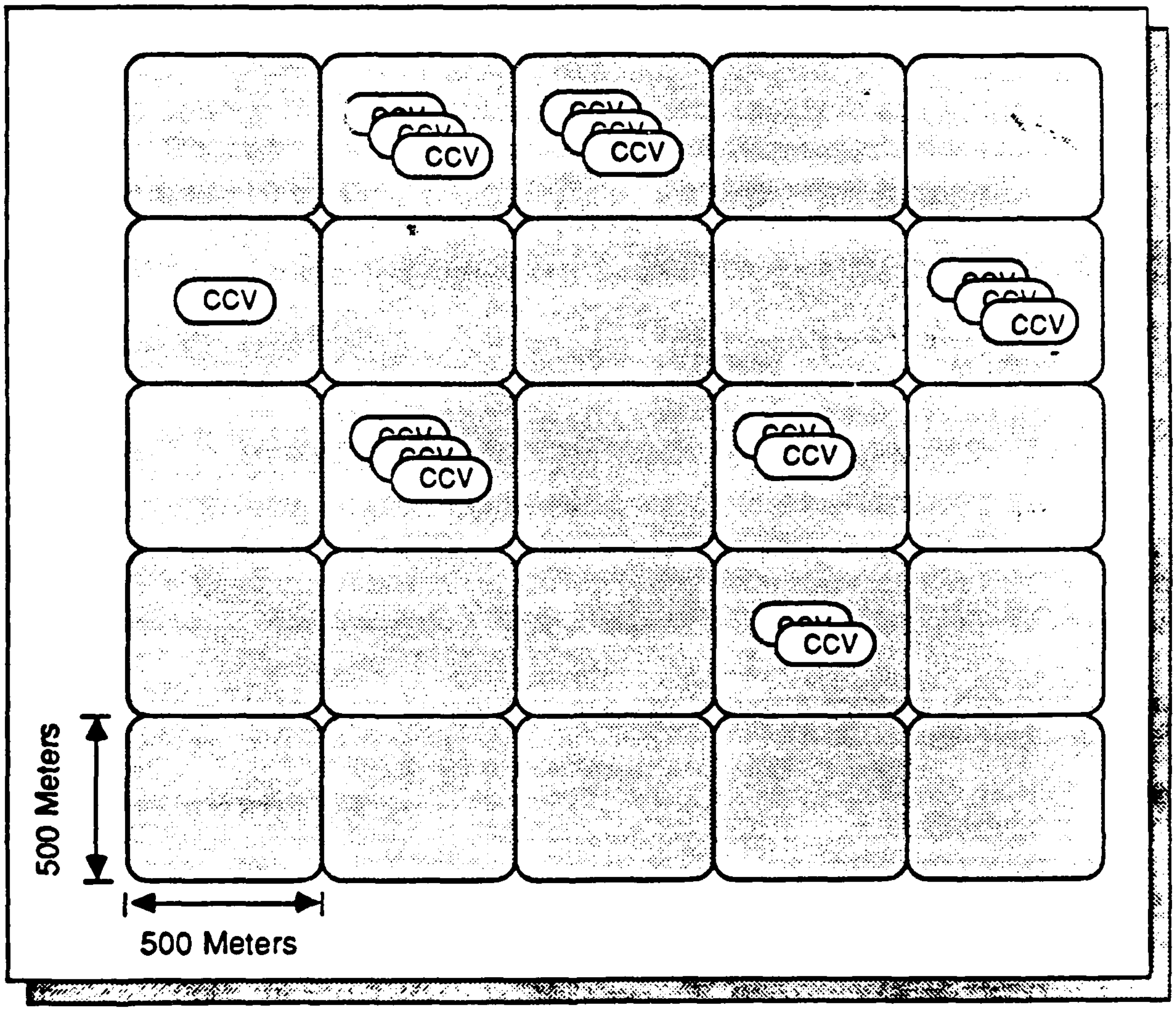


Figure 5-2. CCV Population Density Map.

When the Mother process needs to know which CCVs are near a given point, it determines which bucket that point lies in, and then considers only the CCVs in that bucket (and possibly the ones immediately adjacent to it). In this way, the Mother process needs to consider few CCVs when determining if any are destroyed by indirect fire or present an obstacle to the placement of a vehicle.

5.1.4. Vehicle Placement

The Mother process places both combat vehicles and computer–controlled vehicles on the battlefield at the behest of console processes. By sending a message to the Mother process, a console process can request that a vehicle be placed at a specific location. There are, however, a number of reasons why it may not be possible for the vehicle to appear at the exact location requested:

- the location may already be occupied by another vehicle or by some terrain obstacle, such as a house;
- the location may be too steep to be navigable;
- the soil at the location may be too soft; or
- the location may not be within the bounds of the terrain database being used for the exercise.

The Mother process checks to see if the requested location is suitable. If it is not, the Mother process searches for a suitable spot as close to the original location as possible. The procedure for determining a location's suitability and searching for alternative locations is referred to as *vehicle placement*. This procedure is described in the following paragraphs. With minor exceptions, the Mother process uses the same procedure to place either a simulated combat vehicle or a CCV. Exceptions will be noted as they arise.

The Mother process first determines if the requested location is within the bounds of the terrain database. If the vehicle being placed is a CCV and the requested location is not on the terrain, the CCV enters the disappearing state (described in section 5.1.3) and is not placed. A combat vehicle, however, is always placed. If the location requested for a combat vehicle is not on the terrain, it will be changed to the location on the terrain that is closest to the requested location.

When placing a supply truck or maintenance team truck, the Mother process looks for combat vehicles that are within a 200 meter radius of the requested location. If any combat vehicles are found within this radius, the requested location is changed to the location of the nearest combat vehicle.

After these initial adjustments to the requested location, the Mother process checks the slope and soil type in the area where the vehicle is to be placed. The location is considered unsuitable if it is:

- unpassable muck;
- sloped ground with sandy soil that would give the vehicle a pitch of greater than 26.5 degrees;
- sloped ground with any type of soil that would give the vehicle a pitch of greater than 35 degrees; or
- sloped ground with any type of soil that would give the vehicle a cant of greater than 22 degrees.

Provided that the terrain is not too steep and the soil type is acceptable, the Mother process then checks for terrain objects, such as houses or buildings, that might already occupy the requested location. An object is represented in the terrain database by a quadrilateral footprint and an associated height; this is called a *bounding volume*. To determine if an object interferes with the placement of a vehicle, a rectangle around the quadrilateral footprint of the bounding volume is computed by finding the maximum and minimum north–south and east–west extents of the footprint. This rectangle is called the *extent* of the object. The extent of the vehicle itself is considered to be a 12 by 12 meter square with its center at the requested location. The location is unsuitable if the extent of the vehicle intersects the extent of any object. This method provides a fast and simple way to determine if an object is likely to interfere with the vehicle's placement.

Figure 5-3 gives two examples of this method of collision avoidance. While it is possible that a suitable location might be passed over, as shown in the left half of the figure, the method always guarantees that a vehicle will not collide with any object.

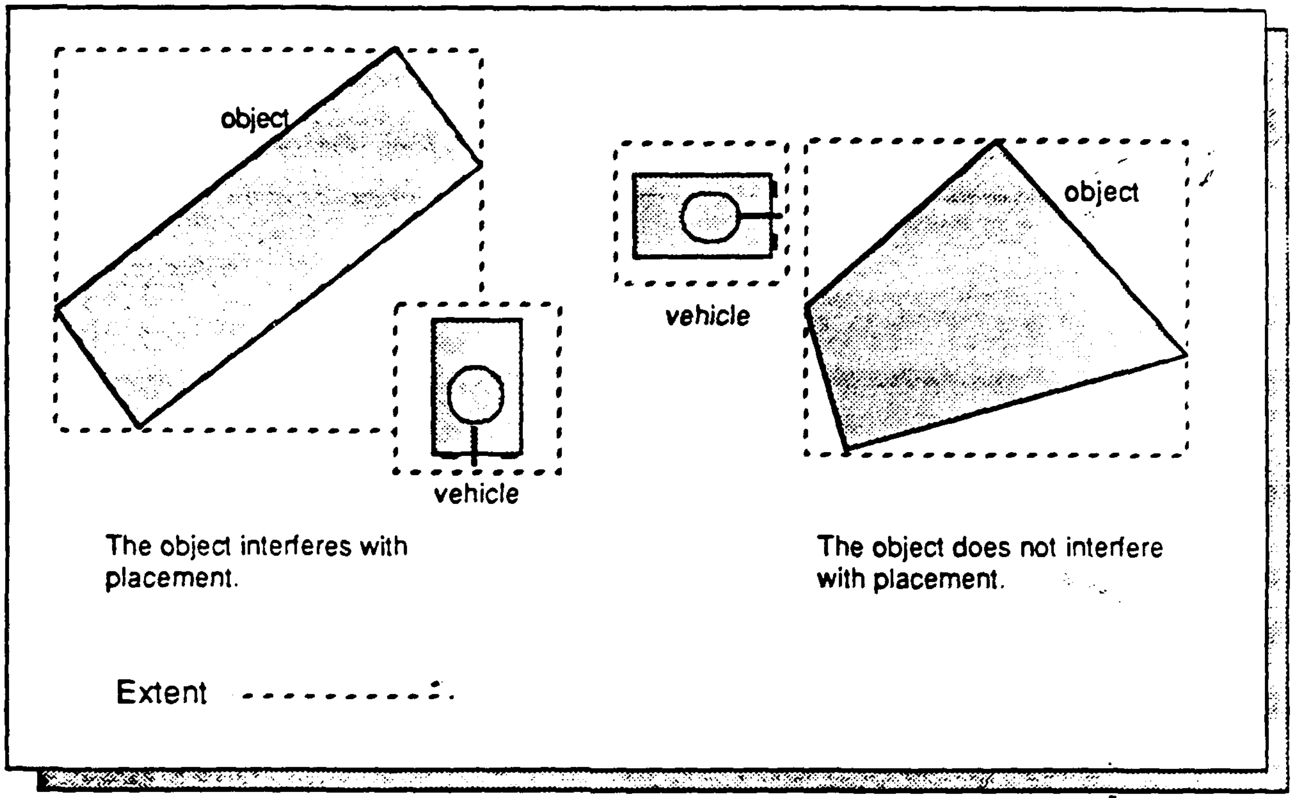


Figure 5–3. Vehicle placement using extents.

Finally, in placing a vehicle, the Mother process checks for other vehicles, already on the terrain, that might be obstacles. For this it uses the same method of intersecting extents described above. If the extent of the vehicle being placed intersects with the extent of another vehicle already on the terrain, then the location is deemed unsuitable.

If the Mother process finds that the requested location is unsuitable because it fails to meet any of the placement criteria, it will try other locations nearby. First, the point 12 meters to the east of the original location is tried; if that location is also unsuitable, the point 12 meters northeast of the original location is tried, then 12 meters north, and so on. If all eight compass points from east around to southwest have been tried and found unsuitable, the trial distance is doubled to 24 meters and a further eight points are tried. The Mother process continues testing locations in this manner until a suitable location is found.

Once a suitable location has been found, if the vehicle being placed is a combat vehicle, the Mother process constructs an Activate PDU and sends it to that vehicle's simulator. If the vehicle being placed is a CCV, the CCV's state is changed to healthy (see section 5.1.3), and an entry is made for the CCV in the CCV Population

Density Map. In either case, the Mother process returns an *ActivateComplete* message to the process requesting the placement, indicating the location at which the vehicle was actually placed.

5.1.6. Bomb and Artillery Shell Detonations

When the CAS process determines that bombs should detonate on the battlefield, or when the FSE process determines that artillery shells should explode on the battlefield, it sends a message to the Mother process. The Mother process computes the locations of the individual detonations, and broadcasts one or more IndirectFire PDUs for them on the SIMNET local area network. Because both artillery shell detonations and bomb detonations are treated in the same manner by the Mother process, they will considered together and called, simply, *detonations* in this section.

The message sent by the FSE and CAS processes describing detonations is a *PlaceBursts* message containing:

- the number of detonations involved;
- what type of ammunition is detonating;
- the north–south and east–west coordinates of each detonation;
- the time delay between each successive detonation; and
- the height of the detonations above the battlefield.

Ignoring any detonations outside the bounds of the terrain database, the Mother process examines the terrain database for each detonation to determine the absolute height of the detonation. It then broadcasts information about the detonations on the SIMNET local area network.

The Mother process also checks for the destruction of CCVs by detonations. It uses the CCV Population Density Map, described in section 5.1.3, to find all CCVs near the detonations. The distance between the location of each detonation and each CCV is calculated and used to assess any damage. When any CCV is destroyed, the Mother process notifies the console process that established the CCV by sending it a *KilledCCV* message.

5.1.7. Exercise Termination

When an exercise is ended by the BattleMaster using the SIMNET Control Console, the SCC process notifies the Mother process by sending it a *Shutdown* message. The Mother process, upon receiving this message, deactivates all of the combat vehicle simulators in the exercise by broadcasting Deactivate PDUs for each simulator on the

SIMNET local area network. It then terminates the other MCC host processes, records statistics collected from the exercise, removes the interprocess communication objects, and terminates.

5.2. Listen Process

The Listen process monitors the SIMNET local area network to detect events of interest to the MCC system, and to maintain a representation of the current situation on the battlefield. When the Listen process receives a PDU signalling an event of interest to the MCC system, it sends a message to the interested process. When it receives a PDU describing the status of a vehicle on the battlefield, it records that information in the Vehicle Table shared memory.

The following sections discuss the types of PDUs that are of interest to the MCC system, and the action taken by the Listen process upon receipt of each.

5.2.1. Vehicle Impact

When a simulated combat vehicle hits another combat vehicle or a CCV with direct fire, the attacking simulator broadcasts a VehicleImpact PDU. This PDU contains the vehicle identifier of the vehicle hit. The Listen process checks all VehicleImpact PDUs received for instances of CCVs being hit. When a CCV is hit, the Listen process sends a *KilledCCV* message to the console process that established the CCV. It also sets a flag in the Vehicle Table shared memory to notify the Mother process that the CCV has been destroyed.

5.2.2. Collision

When a simulated combat vehicle collides with another combat vehicle or CCV, the colliding simulator broadcasts a Collision PDU. This PDU contains the vehicle identifiers of both vehicles involved in the collision. The Listen process checks all Collision PDUs received for instances of CCVs being involved in collisions. When a CCV is hit, the Listen process sends a *KilledCCV* message to the console process that established the CCV. It also sets a flag in the Vehicle Table shared memory to notify the Mother process that the CCV has been destroyed.

5.2.3. Service Request

When a combat vehicle is within 30 meters of a simulated fuel truck, ammunition truck, or maintenance team's truck, the combat vehicle's simulator will periodically broadcast a ServiceRequest PDU. This PDU contains the vehicle identifiers of both the truck and the combat vehicle. The Listen process checks all ServiceRequest PDUs received for cases involving a CCV modeled by the MCC system. When it finds such a PDU, it sends a *ServiceRequest* message to the console process that established the CCV.

5.2.4. Resupply Received

When a simulated combat vehicle has received fuel or ammunition from a supply truck, the combat vehicle's simulator broadcasts a ResupplyReceived PDU. This PDU contains the vehicle identifiers of both the combat vehicle and the supply truck. The Listen process checks all ResupplyReceived PDUs received for cases involving a CCV modeled by the MCC system. When it finds such a PDU, it sends a *VehicleResupplied* message to the console process that established the CCV.

5.2.5. Vehicle Appearance

When the Listen process receives a VehicleAppearance PDU describing a combat vehicle's appearance on the battlefield, it records the location and orientation of that vehicle in the Vehicle Table shared memory.

5.2.6. Vehicle Status

When the Listen process receives a VehicleStatus PDU describing a combat vehicle's supplies and maintenance status, it records this information in the Vehicle Table shared memory.

6. CONSOLES

This chapter describes the implementation of each of the five MCC consoles. We focus here on how the implementation of each console is divided between a host process and a Macintosh application that communicate using a particular set of transactions.

Throughout this chapter, the term *application* is used when referring to the portion of a console's implementation that runs on the console Macintosh; *process* refers to the portion running on the MCC host. The two communicate by means of AppleTalk Transaction Protocol (ATP) transactions. Each transaction involves a request by one party, followed by a response from the other.

6.1. SIMNET Control Console

The SIMNET Control Console is primarily used for initializing components of a simulation exercise, reconstituting vehicles, and placing static vehicles on the battlefield. Its implementation of these features is straightforward.

6.1.1. Initialization

The SIMNET Control Console becomes active as soon as the MCC system is started. It is first used for exercise initialization, during which the console displays a list of terrain databases available for the exercise. This list is supplied to the SCC application on the Macintosh by the SCC process on the host through a series of ATP transactions. When the user completes the dialog for exercise initialization, and commits to the choices made, the SCC application notifies the SCC process with an ATP request. This request contains all of the information entered by the user, including which terrain database was chosen, which companies are participating in the exercise, and which optional elements have been enabled for the exercise.

Following exercise initialization, the SCC process supplies to the SCC application a list of the combat vehicle simulators available for the exercise. This information, which is obtained from the Vehicle Table shared memory, includes the physical location of each simulator, and the type of vehicle simulated by each simulator. The SCC application constructs a simulator table with the information supplied, and uses the table to support simulator allocation and combat vehicle deployment.

When a simulator is assigned to a particular unit, such as a company, that assignment is recorded in the simulator table by the SCC application. Initially, all simulators are recorded as "unassigned", but each can be assigned to any of the units earlier specified as being participants in the exercise.

Once a simulator has been assigned to a unit, the combat vehicle it simulates may be deployed on the battlefield. In deploying the vehicle, the SCC user will specify where the vehicle is to be placed, what direction it should face, and what supplies it should carry. This information is sent, in an ATP request, to the SCC process on the host. That process, in turn, sends a message to the Mother process requesting that the vehicle be placed on the battlefield. Finally, the SCC application notes in its simulator table that the vehicle has been deployed, so that it can prevent any attempt to deploy the same simulator a second time.

Combat service support, fire support, and close air support are also initialized from the SIMNET Control Console. In all three cases, the SCC application only permits the initialization of an element to take place if that element was enabled during exercise initialization. Through a series of dialog boxes, the SCC application collects from the user all of the information needed to initialize the element. This information is then sent to the SCC process on the host, as one or more ATP transactions. The SCC process places the information in the MCC Parameters shared memory, and, when all of the information describing a particular element has been received, it starts the console process responsible for that element.

6.1.2. Reconstitution

From the SIMNET Control Console, the BattleMaster may reconstitute any vehicle on the battlefield. Once he has selected the vehicle he wishes to reconstitute, the SCC application queries the SCC process to obtain the current status of that vehicle. In the case of a combat vehicle, the current status is fetched from the Vehicle Table shared memory. For computer-controlled vehicles, it is fetched from the MCC Parameters shared memory. The vehicle status returned to the SCC application is displayed for the BattleMaster, so that he may view the vehicle's current status while reconstituting it. He may then change any component of the vehicle's status, such as its location. Finally, when the BattleMaster requests it, the SCC application sends the new, modified vehicle status to the SCC process. If a combat vehicle is being reconstituted, the SCC process sends a message to the Mother process requesting that the vehicle be re-initialized on the battlefield. Otherwise, when a computer-controlled vehicle is being reconstituted, the SCC process sends a message to the console process that established the CCV.

6.1.3. Static Vehicles

The SCC application maintains a table of static vehicles on the Macintosh. Each entry describes a single static vehicle, including its location and orientation, its type, and whether it has been destroyed. When the BattleMaster creates a static vehicle, an entry for that vehicle is placed in the table. A description of the static vehicle is also sent, as an ATP request, to the SCC process, which uses the description to make a CCV visible on the battlefield. Whenever the static vehicle is changed—for example, moved to another location—the table entry is updated, and the

SCC process is notified with another request. When the static vehicle is deleted, the table entry is removed, and the SCC process is notified to remove the corresponding CCV from the battlefield.

If a static vehicle is destroyed by direct fire, indirect fire, or collision, the SCC process on the host is notified with a message from the Mother process or the Listen process. The SCC process then notifies the SCC application by sending it an ATP request. Upon receiving this request, the SCC application records, in its static vehicle table, that the vehicle has been destroyed.

6.2. Admin/Log Console

The Admin/Log Console is used to control a fleet of ten ammunition supply trucks, and twelve fuel supply trucks. The trucks can move about the battlefield, transfer supplies to combat vehicles, and take on supplies while at the depots. The trucks also can be destroyed like any CCV, will suffer occasional breakdowns, and can be reconstituted from the SIMNET Control Console.

6.2.1. Data Structures

A table of information about each truck is maintained by both the Admin process on the host, and the Admin/Log application on the Macintosh. These tables record the location, load, company assignment, and state of each truck. Travel by the trucks is modeled on the Macintosh, so the table used by the Admin/Log application also includes the origin, destination, and estimated time of arrival of a truck en route. The host process implements the resupply protocol with combat vehicle simulators, so its table includes the vehicle identifier of any combat vehicle being resupplied.

When the Admin/Log Console is activated, the host process uses a series of ATP transactions to supply the Macintosh application with the initial data for its table. Thereafter the two tables are kept consistent through cooperation between the process and the application. When the host process observes an event causing a change in the status of a truck, it sends an ATP request to the Macintosh application, and vice versa.

6.2.2. Truck Model

Within both the Macintosh application and the host process, each truck is modeled as a system of states, as shown in Figure 6–1. The states are:

ready the truck is stationary on the battlefield, and is available for immediate dispatch

enroute the truck is travelling to some destination *servicing* the truck is supplying a combat vehicle with fuel or ammunition

loading the truck is at the ammunition depot (if an ammunition supply truck) or the fuel depot (if a fuel supply truck).

destroyed the truck has been destroyed

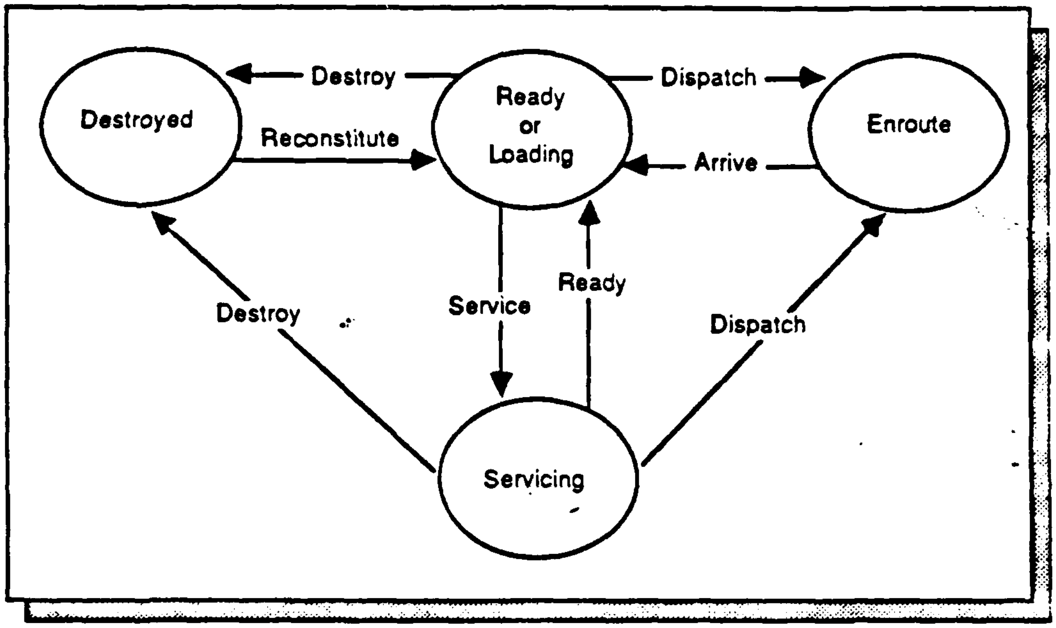


Figure 6–1. Supply truck states.

(The Admin process does not, and has no need to, distinguish between the ready state and the loading state. It considers both to be the ready state.)

These are the transitions between states, and the events that cause them:

dispatch The Admin/Log application will allow a truck to be dispatched only if it is in the ready, the servicing, or the loading state. When a truck is dispatched, the application notifies the host process by sending it an ATP request. The truck enters the enroute state.

arrive The Admin/Log application determines when a truck has arrived at its destination, based on the distance the truck must travel. A truck can also be made to stop travelling, before reaching its destination, by the Admin/Log Console user. When a truck stops travelling, the application notifies the host process by sending it an ATP request. If a truck comes to a halt at its depot, it will be placed in the loading state; otherwise, it will enter the ready state.

service A supply truck transfers supplies to a combat vehicle at the request of that combat vehicle. When the Admin/Log process receives such a request from a combat vehicle simulator, it notifies the application by sending it an ATP request. The supply truck receiving the request enters the servicing state.

ready When the combat vehicle simulator is no longer requesting supplies, the process will notify the application, and the supply truck will return to the ready state or the loading state (depending on whether it is at its depot).

destroy The Admin process will be notified when the truck is destroyed by direct fire, indirect fire, or collision. It sends an ATP request to the application, telling it of the event.

reconstitute When a supply truck is reconstituted from the SIMNET Control Console, the SCC process sends a message to the Admin process. That process, in turn, notifies the application by sending it an ATP request that includes the new status of the truck. The truck is placed in the ready state or the loading state.

6.2.3. Breakdowns

Supply trucks suffer mechanical breakdowns, becoming temporarily disabled at random times during the exercise. This simulation is performed by the Admin/Log application. Any truck not already destroyed can suffer a breakdown. If the truck is in the enroute state, it halts as though the user had stopped it. The breakdown will last an average of fifteen minutes. During that time, the truck can move between ready or loading, servicing, and destroyed states, but it cannot be dispatched. The breakdown is repaired either automatically, after a random period of time, or as a result of the truck being reconstituted.

6.2.4. Servicing

The Admin process implements the SIMNET network protocol procedure used to transfer supplies from a supply truck to a combat vehicle. This procedure is described in *The SIMNET Network and Protocol*[3]. When the procedure begins, the Admin process notifies the application that the truck is servicing, and when the procedure ends, it notifies the application that the truck is ready once more. It also notifies the application of each increment of fuel or ammunition removed from the truck, so the application can maintain a current record of the load on board the truck.

6.3. Maintenance Console

The Maintenance Console is used to control a fleet of ten maintenance teams, represented on the battlefield by their 2-1/2 ton trucks. The teams can move about the battlefield, perform repairs on combat vehicles, and tow combat vehicles to new locations. The teams' trucks also can be destroyed like any CCV, will suffer occasional breakdowns, and can be reconstituted from the SIMNET Control Console. Each team can simultaneously perform, on a single combat vehicle, one repair from each of three different categories. Repairs take from one minute to four hours, depending on the type of repair.

6.3.1. Data Structures

A table of information about each truck is maintained by both the Maint process on the host, and the Maintenance application on the Macintosh. These tables record the location, company assignment, and state of each maintenance team. When a team is servicing a combat vehicle, the tables identify the company and bumper number of the combat vehicle being repaired. Travel by the trucks is modeled on the Macintosh, so the table used by the Maintenance application also includes the origin, destination, and estimated time of arrival of a truck en route. Finally, when a team is towing a combat vehicle, the table entries for that team identify the combat vehicle being towed, and the entry on the host records the supplies and failures status of the combat vehicle. This status information is needed by the host when it re-activates the combat vehicle at the towing destination.

When the Maintenance Console is activated, the host process uses a series of ATP transactions to supply the Macintosh application with the initial data for its table of maintenance teams. Thereafter the tables maintained by the Macintosh application and the host process are kept consistent through cooperation between them. When the host process observes an event causing a change in the status of a maintenance team, it sends an ATP request to the Macintosh application, and vice versa.

The Maintenance application maintains on the Macintosh a table of the repairs in progress, and those completed. An entry in the table, corresponding to one repair, specifies the team performing the repair, the type of repair being performed, the combat vehicle being repaired, and the estimated (or, for a completed repair, actual) time of completion of the repair. Information about repairs is maintained entirely on the Macintosh, and the Maint process is only notified of a repair at the moment it is completed so that it can inform the combat vehicle simulator.

6.3.2. Maintenance Team Model

Within both the Macintosh application and the host process, each maintenance team is modeled as a system of states, as shown in Figure 6-2. The states are:

ready	the team is stationary on the battlefield, and is available for immediate dispatch
enroute	the team is travelling to some destination
servicing	the team is within range of a combat vehicle such that it can perform repairs on it
recovering	the team is within range of a combat vehicle, and is preparing to tow it
towing	the team is travelling to some destination with a combat vehicle in tow
destroyed	the team has been destroyed

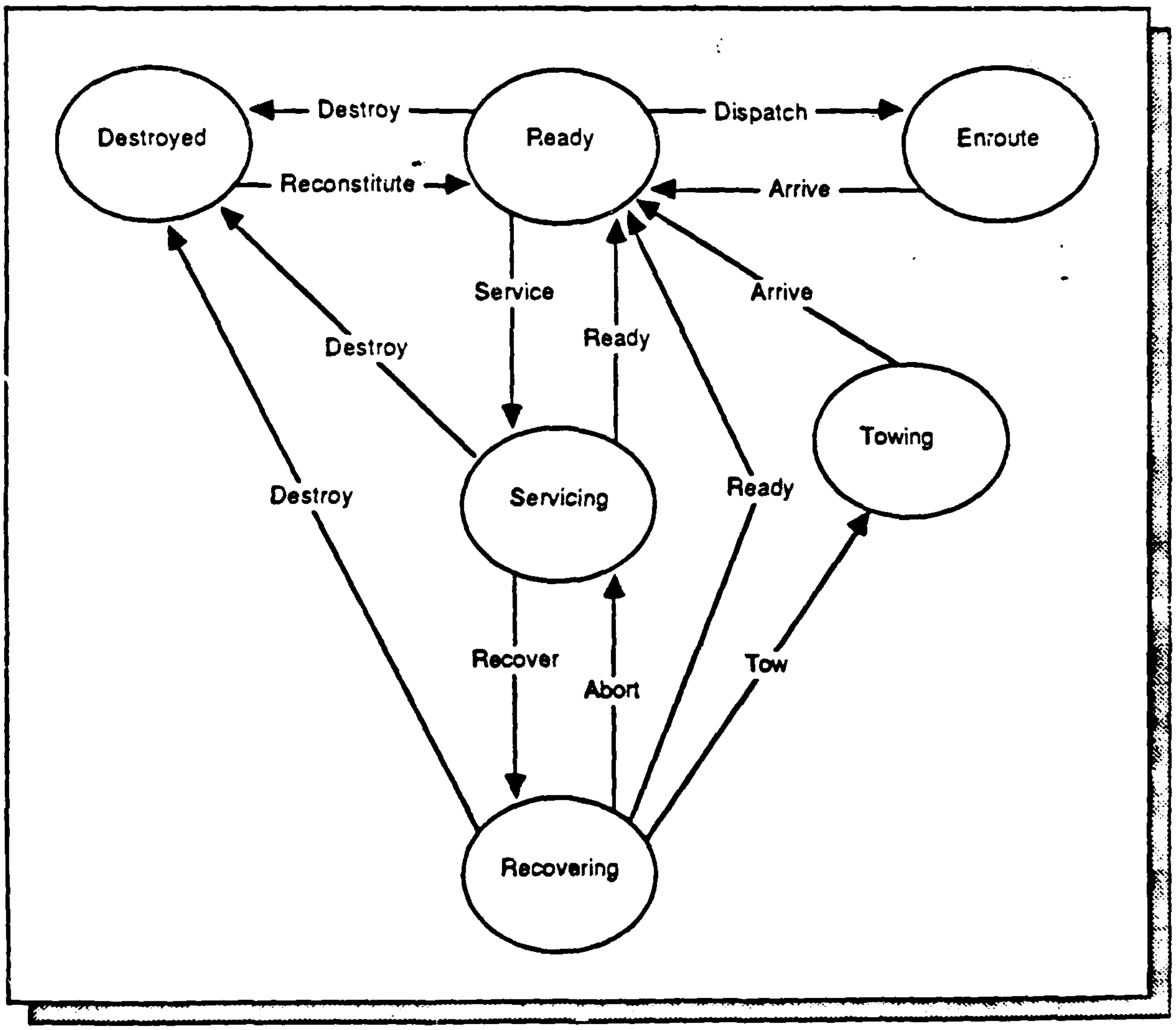


Figure 6-2. Maintenance team states.

(The Maint process does not, and has no need to, distinguish between the servicing state and the recovering state. It considers both to be the servicing state.)

These are the transitions between states, and the events that cause them:

dispatch The Maintenance application will allow a team to be dispatched only if it is in the ready state or the servicing state. When a team is dispatched, the application notifies the host process by sending it an ATP request. The team enters the enroute state.

arrive The Maintenance application determines when a team has arrived at its destination, based on the distance the truck must travel. A team can also be made to stop travelling, before reaching its destination, by the Maintenance Console user. When a team stops travelling, the application notifies the host process by sending it an ATP request, and the team enters the ready state. If the team was towing a combat vehicle, the Maint process will re-activate the combat vehicle being towed.

service A combat vehicle simulator has noticed that the maintenance team is nearby, and has engaged it using the vehicle repair protocol procedure. It does so by sending ServiceRequest PDUs to the MCC system. When the Maintenance process receives these requests, it notifies the application by sending it an ATP request. The team enters the servicing state.

ready When the combat vehicle simulator is no longer requesting service from the team, the process will notify the application, and the team will return to the ready state.

recover While a team is in the servicing state, the Maintenance Console user can initiate towing of the combat vehicle being serviced. When he does so, the Maintenance application places the team in the recovering state and starts a five minute timer. Five minutes represents the time required by the maintenance team to prepare for towing.

tow When the five minute timer expires, the Maintenance application notifies the Maint process that towing is beginning. The Maint process makes the team's truck disappear from the battlefield, and deactivates the combat vehicle simulator. It records, from the Vehicle Table shared memory, the most recent supplies and failures status of the combat vehicle so that the vehicle can be re-activated with the same status. The team enters the towing state.

abort While the team is preparing to tow a combat vehicle, the Maintenance Console user can revoke his towing request. The team returns to the servicing state if he does so.

destroy The Admin process will be notified when the truck is destroyed by direct fire, indirect fire, or a collision. It sends an ATP request to the application, telling it of the event.

reconstitute When a supply truck is reconstituted from the SIMNET Control Console, the SCC process sends a message to the Admin process. That process, in turn, notifies the application by sending it an ATP request that includes the new status of the truck. The truck is placed in the ready state or the loading state.

6.3.3. Breakdowns

Maintenance teams' trucks suffer mechanical breakdowns, becoming temporarily disabled at random times during the exercise. This simulation is performed by the Maintenance application. Any team not already destroyed can suffer a breakdown. If the team is in the enroute state, it halts as though the user had stopped it. The breakdown will last an average of fifteen minutes. During that time, the team can move between ready and servicing states, and it can perform repairs, but it cannot be dispatched. The breakdown is repaired either automatically, after a random period of time, or as a result of the team being reconstituted.

6.3.4. Repairs

The Maint process implements the SIMNET network protocol procedure used to repair a combat vehicle. This procedure is described in *The SIMNET Network and Protocol*[3]. When a combat vehicle simulator notices a nearby maintenance team, it begins periodically sending ServiceRequest PDUs to the MCC system. These PDUs are passed to the Maint process by the Listen process. As long as the Maint process continues to receive these PDUs, it allows the team to remain in the servicing state or the recovering state.

While a team is in the servicing state, the Maintenance application will allow the team to begin and carry out repairs. Any time the team leaves the servicing state, any repairs it has in progress are abandoned. However, when a repair runs to completion, the Maint process is notified of the repair with an ATP request. The Maint process then tells the simulator of the repair by sending it a Repair PDU.

6.4. Fire Support Console

The Fire Support Console is the most complex of the MCC consoles. It models a platoon of mortars and a battery of howitzers, and allows a variety of indirect fire missions to be performed with those guns. The console supports the fire support officer's planning process by allowing him to prepare a list of targets, and schedule indirect fire missions in advance. The console also allows the FSO to move individual sections of the mortar platoon on the battlefield.

In the discussion that follows, we use the term *gun* to mean either a self-propelled howitzer, or a mortar carrier. A *gun group* is the smallest collection of these that the FSO can explicitly control: for the howitzers, that's a platoon of four, and for the mortars, it's a section of three.

6.4.1. Howitzers and Mortars

The FSE application on the Macintosh models gun groups and the indirect fire missions those groups are engaged in, but it does not model the individual guns. For each gun group, the application records:

- the location of the group's center of mass, and its azimuth of fire;
- the number of guns within the group that are operational;
- the quantity of ammunition available to the group;
- the state of the group: ready, busy firing a mission, enroute across the battlefield, or setting up after travelling;
- if the group is firing a mission, the number of rounds fired so far;
- if the group is firing a mission, the time when it will be ready to fire another round;
- if the FSO has scheduled a movement of the group, the time the movement is to start, and the destination of the movement; and
- if the group is currently moving, the time the group will arrive at its destination.

The FSE process on the host supplies the locations, azimuths of fire, and quantities of ammunition to the FSE application on the Macintosh at the time the console is activated. It also provides this information whenever a gun group is reconstituted by the BattleMaster.

The FSE process on the host models the individual guns, but has no concept of the indirect fire missions they perform. When a gun group is deployed on the battlefield, the FSE process computes the position of each gun, and renders a CCV visible for that gun. When a gun is destroyed by indirect fire, direct fire, or a collision, the FSE process notifies the FSE application so that the application can adjust its count of the number of guns operational in the appropriate gun group.

6.4.2. Indirect Fire Missions

The FSE application on the Macintosh maintains a table with room for four indirect fire missions. Each mission is described by:

- the call sign of the observer requesting the mission, as entered by the FSO;
- the type of mission: a regular indirect fire mission, or a final protective fire mission;
- the state of execution of the mission: the gun groups are quiescent (awaiting further orders related to the mission), awaiting a command to start firing, performing an adjustment, or performing a fire for effect;
- which gun groups have been selected by the FSO to perform the mission;
- the location of the target to be fired upon;
- the type of ammunition and fuze to fire;
- if a fire for effect is being performed, the number of rounds to fire; and
- if a fire for effect or final protective fire is being performed, the time when a gun group firing the mission will be ready to fire another round.

A mission is added to the table when the FSO requests a new mission, and removed from the table when he indicates that a mission is complete. A mission is also added when a mission scheduled in advance comes due, and that mission is removed when the scheduled number of rounds have been fired.

When the FSO gives an order for the gun groups performing a mission to fire, the FSE application begins simulating the firing of a series of rounds. With each round fired, the application sends a transaction to the FSE process telling it which gun groups fired, what kind of ammunition they fired, where the target is located, and what kind of mission they are firing. When it receives this transaction, the FSE process modifies the appearances of the appropriate guns, causing them to display muzzle flashes on the battlefield. The FSE process then delays for a period corresponding to the time of flight of the shells. Following this delay, the FSE process computes the pattern of detonations on the battlefield, and requests the Mother process to broadcast an IndirectFire PDU describing those detonations.

After each gun group fires a round, the FSE application on the Macintosh decrements the quantity of ammunition recorded for that group. Then, based on the type of guns and the number of rounds they have fired already, it computes the time when they will be ready to fire another round. The application periodically checks these times to determine when to fire another round from a gun group.

6.4.3. Target Lists

Using the Fire Support Console, the FSO can prepare lists of indirect fire targets and final protective fire targets. He assigns to each target a number. Later, when requesting a fire mission, he can specify the target by entering one of these numbers.

The target lists are maintained entirely on the Macintosh by the FSE application. When the FSO specifies a target by number, the FSE application simply substitutes the known UTM grid coordinate of that target.

6.4.4. Displacement

Movement of the mortar sections about on the battlefield is modeled by the FSE application. It allows the FSO to schedule a time for a mortar section to move, and, based on the distance moved, estimates a time for the section to arrive at its destination. When the section begins its move, the application notifies the FSE process, and that process causes the individual guns of the section to vanish from the battlefield. When the section arrives at its destination, the process is again informed. The FSE process deploys the individual guns, making them visible once more.

For five minutes after a mortar section has arrived at its destination, it is "setting up" and cannot accept fire missions. This period is timed, and its restrictions enforced, by the FSE application.

6.5. Close Air Support Console

The Close Air Support Console maintains a schedule of past, present, and future close air support missions. The air liaison officer using that console can examine this schedule, add new missions to it, and modify the parameters of present and future missions.

When the close air support simulation is initialized at the SIMNET Control Console, limits are established for the number of sorties available during the current 24-hour period. These limits are passed to the CAS process via the MCC Parameters shared memory, and passed from there to the CAS application when it is activated. The application enforces these limits directly as new missions are added to the schedule.

The schedule of missions is implemented and represented entirely within the Macintosh application. That application examines the schedule frequently for missions that have arrived "on station", or reached their "bingo fuel" time, to notify the ALO of these condition A mission can be "cleared hot" by the ALO between the time it arrives on station, and the time, ten minutes later, when it reaches bingo fuel.

Only at the moment a mission is "cleared hot" does the application actually notify the CAS process on the host of the mission, by means of an ATP request. The request bears the target location, the run-in heading, and the number of sorties participating in the mission. The CAS process uses this information to compute the locations of individual bomb detonations on the battlefield. Then it sends a message to the Mother process requesting that the detonations be reported on the SIMNET local area network. If the mission involves more than one sortie, the CAS process repeats its request to the Mother process, once for each sortie, at intervals of five seconds.

From the SIMNET Control Console, the BattleMaster may make additional sorties available for close air support. When he does so, notice of the additional sorties is passed from the SCC application to the SCC process by transaction, from there to the CAS process by message, and finally, from the CAS process to the CAS application by transaction.

7. REFERENCES

[1] "The Division 86 Tank Battalion/Task Force", Field Manual Number 17-17. Department of the Army. Washington, D.C., 1983.

[2] "SIMNET Master Documentation". Perceptronics, Inc. 1986.

[3] "The SIMNET Network and Protocol", BBN Report Number 6369. BBN Laboratories, Inc. Cambridge, Mass., 1987.

[4] Firing Table FT 155-AN-1. Department of the Army. Washington, D.C.

[5] Firing Table FT 4.2-H-2. Department of the Army. Washington, D.C., 1968.

[6] Apple Computer, Inc. "Inside Macintosh", Volumes 1-4. Addison-Wesley Publishing Company, Inc. Reading, Mass., 1985, 1986.

[7] Apple Computer, Inc. "Inside AppleTalk". 1986.

[8] "SIMNET M1 Abrams Main Battle Tank Simulation: Software Description and Documentation", BBN Report Number 6323. BBN Laboratories, Inc. Cambridge, Mass., 1987.

APPENDIX A. MCC CONFIGURATION FILE

The MCC system reads a file of configuration information when it is started. This appendix describes the format of that file.

Each line, or *entry*, in the configuration file consists of a keyword followed by one or more parameters. There are six different keywords recognized: *vehicle*, *terrain*, *console*, *logfile*, *aircraft*, and *bridge*. These may appear in the file in any order, but each must be on a separate line. Parameters follow the keyword on the same line, separated by one or more blanks and/or tabs. Each entry is described below.

A.1. Vehicle

The *vehicle* entry describes a vehicle simulator to the MCC system. The configuration file should contain a vehicle entry for each simulator that is available for use in simulation exercise. The parameters of the vehicle entry are, in order, *address*, *trailer*, *trailer element*, and *type*. Address is the simulator's address on the SIMNET local area network. Trailer and trailer element specify the physical location of the simulator; they are a number and a letter, respectively. Type is the type of vehicle that the simulator simulates.

Format:	vehicle	<address>	<trailer>	<trailer element>	<type>
Sample Entry:	vehicle	02cf1f100786	1	A	M1

A.2. Terrain

The *terrain* entry describes a terrain database stored on the MCC host's disk drive. The configuration file should contain one terrain entry for each terrain database available to the MCC system. The terrain entry has one parameter, *directory*, which is the name of a directory on the host's filesystem containing the terrain database files.

Format:	terrain	<directory>
Sample Entry:	terrain	/fs/a/terrain/FtKnox

A.3. Console

The *console* entry specifies the name the MCC host should use when trying to locate a console Macintosh with the AppleTalk Name Binding Protocol. The configuration file should contain a console entry for each console of the MCC system. The console entry has two parameters: *type*, and *name*. Type is the type of the MCC console: *scc* (for SIMNET Control Console), *al* (for Admin/Log Console), *maint* (for Maintenance Console), *fse* (for Fire

Support Console), or *cas* (for Close Air Support Console). Name is the name of that console's Macintosh. Because a console name may be more than one word, it must be enclosed in quotation marks.

Format:	console	<type>	"<name>"
Sample Entry:	console	al	"Admin/Log"

A.4. Logfile

The *logfile* entry describes a file in which the MCC system records the starting time, ending time, and statistics compiled for each simulation exercise. This entry has one parameter, *file*, which is the name of the file to be used by the MCC system.

Format:	logfile	<file>
Sample Entry:	logfile	/simnet/data/logfile

A.5. Aircraft

The *aircraft* entry provides the MCC system host with a number of parameters used to model close air support aircraft. This entry has five parameters: *speed, bombs/sortie, scatter-x, scatter-y,* and *aircraft spacing*. Speed is the air speed of a bomber, and is used to determine the time between successive bomb detonations. Bombs/sortie is the number of bombs dropped by each close air support aircraft. Scatter-x and scatter-y are used to calculate the pattern of the bomb detonations on the battlefield. Aircraft spacing is the time, in seconds, between attacks of successive sorties in the same mission.

Format:	aircraft	<speed>	<bombs/sortie>	<scatter-x>	<scatter-y>	<aircraft spacing>
Sample Entry:	aircraft	100	12	25	5	5

A.6. Bridge

The *bridge* entry tells the MCC host where to find the Bridge linking it to the AppleTalk network. It has one parameter, *device*, which gives the name of the RS-232 port to which the Bridge is connected.

Format:	bridge	<file>
Sample Entry:	bridge	/dev/tty10

APPENDIX B. INTERPROCESS MESSAGE SUMMARY

This chapter describes each of the messages exchanged by processes on the MCC host. The data included with each message, and the circumstances under which each is sent, are also described.

B.1. ShowCCV

The *ShowCCV* message is sent to the Mother process by the SCC, FSE, Admin, and Maint processes when one of these processes wants a CCV to become visible on the terrain. The following data are included:

- the vehicle identifier of the CCV to be shown;
- the type of vehicle (M1, T72, mortar carrier, etc.);
- the location at which the CCV is to appear on the battlefield; and
- the direction the CCV is to face.

This message is the first part of a round-trip interaction whose reply is an *ActivateComplete* message.

B.2. HideCCV

The *HideCCV* message is sent to the Mother process by the SCC, FSE, Admin, and Maint processes when one of these processes wants a CCV to become invisible. The vehicle identifier of the CCV is included in the message.

This message involves a one-way interaction.

B.3. KilledCCV

This message is sent by the Listen process to the Admin, Maint, SCC, and FSE processes when a VehicleImpact PDU or Collision PDU is received from the SIMNET network indicating that a CCV established by one of these processes has been destroyed. It is sent by the Mother process when a detonation included in an IndirectFire PDU being broadcast by that process destroys a CCV. The vehicle identifier of the destroyed CCV is included.

This message involves a one-way interaction.

B.4. TruckReconstitute

The *TruckReconstitute* message is sent to the Admin process or the Maint process when a supply truck or maintenance team is reconstituted. The data included are:

- a number identifying the supply truck or maintenance team;
- the type of the vehicle being reconstituted;
- the new location of the reconstituted vehicle, if appropriate;
- the supplies on board the reconstituted vehicle;
- the company to which the vehicle has been assigned; and
- the force to which the vehicle has been assigned (offense, defense, or shared).

This message involves a one-way interaction.

B.5. ActivateVehicle

The *ActivateVehicle* message is sent to the Mother process by the SCC process when it wants to activate a combat vehicle simulator. It is also used by the Maint process to re-activate a combat vehicle when it has reached its towing destination. The following information is included in the message:

- the vehicle identifier of the simulator to be activated;
- the type of simulator (e.g., M1);
- the location at which the simulated vehicle should be placed;
- the desired orientation of the vehicle's hull and turret; and
- the supplies and failures status of the vehicle.

This message is the first part of a round-trip interaction whose reply is an *ActivateComplete* message.

B.6. ActivateComplete

This message is sent to the SCC, Admin, Maint, or FSE process by the Mother process after it has placed a combat vehicle or CCV on the battlefield. The vehicle identifier and actual location of the vehicle or CCV are included.

This message is the second part of a round-trip interaction whose first part is a ShowCCV message or an ActivateVehicle message.

B.7. DeactivateVehicle

The *DeactivateVehicle* message is sent to the Mother process by the Maint process when a combat vehicle simulator is about to be towed. The vehicle identifier of the simulator to be deactivated is sent.

This message involves a one-way interaction.

B.8. PlaceBursts

The *PlaceBursts* message is sent to the Mother process by the FSE process at the moment some artillery shells are to detonate on or above the battlefield. It is also sent to the Mother process by the CAS process at the moment some bombs are to explode on the battlefield. The following data are included:

- the type of ammunition detonating;
- the number of shells or bombs exploding; and
- for each shell or bomb, the horizontal location of its detonation, the height of its detonation above the terrain, and the delay between its detonation and that of the succeeding one.

This message involves a one-way interaction.

B.9. Sorties

The *Sorties* message is sent by the SCC process to the CAS process when close air support sorties have been allotted by the BattleMaster. The message includes the number of sorties allotted, and the number of those that may be preplanned.

This message involves a one-way interaction.

B.10. ServiceRequest

The *ServiceRequest* message is sent by the Listen process to the Admin process or the Maint processes when a ServiceRequest PDU is received from a combat vehicle simulator. The PDU signifies a request for fuel or ammunition from a supply truck, or a request for maintenance from a maintenance team truck. The message is sent to the process that established the truck to which the request is being made, and it includes the vehicle identifiers of the truck and the combat vehicle.

This message involves a one-way interaction.

B.11. VehicleResupplied

This message is sent to the Admin process by the Listen process when a ResupplyReceived PDU is received from a combat vehicle simulator, meaning that it has accepted fuel or ammunition from a supply truck. Included in the message are:

- the vehicle identifier of the vehicle receiving supplies;
- the vehicle identifier of the supply truck providing supplies;
- the amount of fuel taken, if any; and
- the amount of each type of ammunition taken, if any.

This message involves a one-way interaction.

B.12. ATalkSend

The *ATalkSend* message is sent to the ATSend process by any other process that wishes to send a Datagram Delivery Protocol (DDP) packet on the AppleTalk network. The message includes:

- the DDP socket number of the sending process;
- the node number and DDP socket number of the datagram's destination;
- the length of the data portion of the DDP packet;
- a pointer to the buffer containing the data to be sent, in the AppleTalk Info shared memory; and

- a flag indicating whether the buffer is to be released once the packet has been sent.

This message involves a one-way interaction.

B.13. ATalkRecv

The *ATalkRecv* message is sent by the ATRecv process to a console process when a DDP packet is received that is addressed to a DDP socket owned by the console process. The message includes:

- the DDP socket number of the receiving process;
- the node number and DDP socket number of the sender;
- the length of the data portion of the DDP packet; and
- a pointer to the buffer containing the data received, in the AppleTalk Info shared memory.

This message involves a one-way interaction.

B.14. RestartConsole

The *RestartConsole* message is sent by the Terminal process to the SCC process when a command is entered at the host terminal to restart a console. The message includes the number of the console process that is controlling the console to be restarted. This message involves a one-way interaction.

B.15. ResetClock

The *ResetClock* message is sent by the Terminal process to the SCC process when a command is entered at the host terminal to set the MCC system's clock. The SCC then sends the message to the Maint, Admin, CAS, and FSE processes so that they can each inform their respective Macintosh applications. This message involves a one-way interaction.

B.16. InitTerrain

The *InitTerrain* message is sent by the SCC process to the Mother process when exercise initialization has been completed at the SIMNET Control Console. The message specifies the terrain database chosen for the exercise. This message involves a one-way interaction.

B.17. Shutdown

The *Shutdown* message is sent to the Mother, Admin, Maint, FSE, and CAS processes by the SCC process when an exercise is ended by the BattleMaster. The message is also used by the SCC process when restarting a console in response to a RestartConsole message from the Terminal process. This message involves a one-way interaction.